CP/NET
Network Operating System
Reference Manual

Digital Research
P.O. Box 579
160 Central Avenue
Pacific Grove, CA 93950
(408) 649-3896
TWX 910 360 5001

[CP/M3 extensions added April, 2016]

COPYRIGHT

DISCLAIMER

TRADEMARKS

The CP/NET Network Operating System Reference Manual was prepared using the Digital Research TEX Text Formatter and printed in the United States of America by Commercial Press/Monterey.

Fifth Edition: November 1982

Foreword

CP/NET®, a network operating system, enables microcomputers to access common resources via a network. CP/NET allows microcomputers to share and transfer disk files, to share printers and consoles, and to share programs and data bases. CP/NET consists of servers running MP/M II® and requesters running CP/M®. The servers are hosts that manage the shared resources that the network requesters can access.

The hardware environment for CP/NET must include two or more microcomputers that can communicate in some way.

One of the microcomputers must execute the MP/M II operating system to provide the CP/NET server facilities. The processor executing MP/M II must be an 8080, 8085, or Z80 CPU with a minimum of 32K bytes of memory, 1 to 16 consoles, 1 to 16 logical or physical disk drives each containing up to eight megabytes, a clock/timer interrupt, and a network interface.

The CP/NET requester microcomputers must have 8080, 8085, or Z80 CPUs with at least 16K bytes of memory, 0 to 16 logical or physical disk drives each containing up to eight megabytes, and a network interface. A console is not absolutely required although it is strongly recommended.

The CP/NET Network Operating System Reference Manual is intended for several different levels of CP/NET users. It contains all the information you need to use CP/M applications programs on a CP/NET requester, to write new application programs under CP/NET, and to customize CP/NET for a specific network.

Section 1, an overview of the CP/NET system, discusses CP/NET features, network topologies, and the principles behind CP/NET operation.

Section 2 contains all the information you need to use the network when executing CP/M application programs. You need no skill level beyond that required for normal CP/M operation.

Section 3 describes the CP/NET interprocessor message format and each of the Network Disk Operating System (NDOS) functions you can invoke from application programs. This section provides the information you need to access the network primitives. Section 3 also discusses the implications of performing CP/M operations on a resource controlled by the MP/M II operating system.

Section 4 provides information for the systems programmer. This section describes how to write a custom Slave Network 1/0 System (SNIOS) that performs the CP/NET requester network functions. The mechanics of implementing and debugging a custom SNIOS are also discussed. Programmers attempting to develop an SNIOS should be familiar with CP/M and experienced in writing a custom CP/M BIOS. This section also explains how to write a custom Network Interface Process (NETWRKIF) that performs the CP/NET server network functions.

Section 4 also discusses implementing and debugging the NETWRKIF module. You must have a high degree of competence and experience with MP/M II to develop a custom NETWRKIF. You must be familiar with the process and queue descriptor data structures and the MP/M II XDOS primitive functions. Experience with implementing an XIOS for MP/M II might also be necessary.

Appendixes to this manual contain several example network communications packages.

Table of Contents

- 1 CP/NET Overview
  - 1.1 CP/NET Features
  - 1.2 CP/NET Configurations
  - 1.3 How the Requester Works
  - 1.4 How the Server Works
- 2 CP/NET Utilities
  - 2.1 The LOGIN Command
  - 2.2 The LOGOFF Command
  - 2.3 The NETWORK Command
  - 2.4 The LOCAL Command
  - 2.5 The ENDLIST Command
  - 2.6 The DSKRESET Command
  - 2.7 The CPNETLDR Command
  - 2.8 The CPNETSTS Command
  - 2.9 CTRL-P
  - 2.10 The MAIL Utility
    - 2.10.1 Menus
    - 2.10.2 Data Entry
    - 2.10.3 MAIL Options
    - 2.10.4 Error Messages
- 3 CP/NET Programmer's Guide
  - 3.1 CP/NET Interprocessor Message Format
    - 3.1.1 Message Format Code
    - 3.1.2 Message Destination Processor ID
    - 3.1.3 Message Source Processor ID
    - 3.1.4 CP/M Function Code
    - 3.1.5 Size
    - 3.1.6 CP/NET Message
    - 3.1.7 Additional Packaging Overhead
  - 3.2 Running Applications Transparently Under CP/NET
    - 3.2.1 MP/M II vs. CP/M File Systems
    - 3.2.2 Error Handling Under CP/NET
    - 3.2.3 Temporary Filename Translation
    - 3.2.4 Opening System Files on User 0
    - 3.2.5 Compatibility Attributes
    - 3.2.6 Password Protection Under CP/NET
    - 3.2.7 Networked List and Console Devices
  - 3.3 CP/NET Function Extensions to CP/M
  - 3.4 CP/NET Applications
- 4 CP/NET System Guide
  - 4.1 General Network Considerations
    - 4.1.1 Functions of the CP/NET Physical Modules
    - 4.1.2 Interfacing a Computer to a Network
    - 4.1.3 Developing a Network Layer
    - 4.1.4 Error Recovery
  - 4.2 Customizing the Requester's SNIOS
    - 4.2.1 Slave Network 1/0 System Entry Points
    - 4.2.2 Requester Configuration Table

4.2.3 Prefiguring the Configuration Table
4.2.4 Sending and Receiving Messages
4.2.5 Generating and Debugging a Custom SNIOS
4.3 Customizing the Server
4.3.1 Detecting and Receiving Incoming Messages
4.3.2 The Architecture of the NETWRKIF Module
4.3.3 Elements of the NETWRKIF
4.3.4 Enhancements and Additions to the NETWRKIF
4.3.5 MP/M II Performance Factors and NETWRKIF
4.3.6 Generating the NETWRKIF
4.3.7 Debugging the NETWIRKIF
4.4 Implementing Non-MP/M II Servers

Appendixes

A CP/NOS
A.1 Overview
A.2 System Requirements
A.3 Customizing the CP/NOS
A.4 Building the CP/NOS System
A.5 Debugging the System

B CP/NET 1.2 Standard Message Formats

C CP/NET 1.2 Logical Message Specifications

D NDOS Function Summary

E A Simple RS-232C CP/NET System
E.1 Protocol Handshake
E.2 Binary Protocol Message Format
E.3 ASCII Protocol Message Format
E.4 Modifying the SNIOS
E.5 Modifying the NETWRKIF

F A CP/NET System for Use with ULCnet
F.1 Overview of ULCnet
F.2 Customizing a ULCnet SNIOS for the Requester
F.3 Creating the ULCnet Server

G Using CP/NET 1.2 With Corvus OMNINET
G.1 The Corvus Engineering Transporter
G.2 Implementation Structure
G.3 The SNIOS Implementation
G.4 The NETWRKIF Implementation Model
G.5 Possible Improvements to NETWRKIF

Tables

2-1. Receive Mail Message-handling Options

3-1. Interface Attributes
3-2. BDOS Error Modes

4-1. Requester Configuration Table
4-2. Server Configuration Table
4-3. Requester Control Block

B-1. Message Field Length Table

C-1. Conventional CP/NET Messages

D-1. NDOS Functions

G-1. Transporter Command Block
G-2. Receive Result Block

Figures

1-1. Standard CP/NET Configuration
1-2. CP/NOS Configuration
1-3. Single Requester Networked to MP/M II Server
1-4. Multiple Requesters in Hub-star Configuration
1-5. Multidrop Network
1-6. Hybrid Network
1-7. CP/NET Memory Structure
1-8. A Simple Server Supporting Three Requesters

3-1. Message Format

4-1. Layered Model of a CP/NET Network Node
4-2. Network Status Byte Format
4-3. Algorithm for Interrupt-driven Requester Node
4-4. Server Architecture
4-5. Two-process NETWRKIF
4-6. Transport Process/Data-link Processes Interface
4-7. Directly Interfacing NETWRKIF to XIOS Routines
4-8. Synchronizing Data-link Activity Using Flags
4-9. A Typical Server Memory Map
4-10. Implementing Timeouts with Flags

B-1. CP/NET 1.2 Logical Message Format

E-1. Protocol Handshake
E-2. Binary Protocol Message Format
E-3. ASCII Protocol Message Format

Listings

2-1. A Typical CPNETLDR Execution
2-2. A Typical CPNETSTS Execution

4-1. SNIOS Jump Vector
4-2. Stack and Process Descriptor Allocation for Four-requester Server

E-1. Requester Network I/O System

E-2. Server Network I/F Module

F-1. Requester Network I/O System for ULCnet
F-2. NETWRKIF for Systems Running ULCnet
F-3. ULCnet Data-link Layer MP/M XIOS Module

G-1. Sample SNIOS for Corvus OMNINET
G-2. Sample Server Network I/F for Corvus OMNINET

Section 1
CP/NET Overview

By separating the logical operating system from the hardware environment and placing all hardware-independent code in a separate I/O module, CP/M and MP/M II have gained widespread industry acceptance. The CP/NET operating system uses this same design approach. CP/NET is network independent. The Slave Network I/O System (SNIOS) module contains all network-dependent code for the requester. The Network Interface Process (NETWRKIF) module contains all network-dependent code for the server. Logical messages passed to and from the SNIOS or NETWRKIF are transmitted over an arbitrary network between servers and requesters using an arbitrary network protocol.

CP/NET and CP/NOS can be combined in a composite network consisting of MP/M II servers, CP/M requesters, and diskless CP/NOS requesters.

CP/NET is a bridge between a microcomputer running MP/M II and a microcomputer running CP/M. The MP/M II server manages resources that are considered public to the network. The CP/NET requesters executing CP/M have access to the public resources of the server and to their own local private resources, which cannot be accessed from the network. This architecture permits the server's resources to be shared among the requesters, yet guarantees the security of the requester's resources.

The MP/M II server responds to the network asynchronously in real-time; the CP/M requesters perform sequential I/O and are usually not capable of monitoring a network interface in real-time. Figure 1-1 illustrates the relationship between CP/M, MP/M II, and CP/NET.

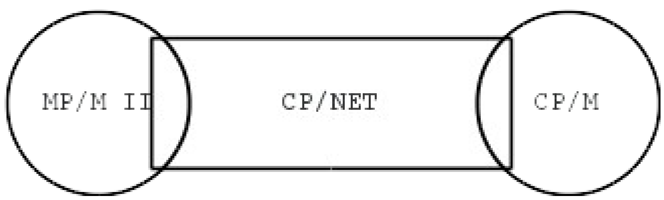


Figure 1-1. Standard CP/NET Configuration

CP/NOS, the second network operating system product, is designed for applications where the requester microcomputer lacks disk resources and is therefore unable to run CP/M. CP/NOS consists of

- a bootstrap loader that can be placed into ROM or PROM
- a skeletal CP/M containing only the console and printer functions
- the logical and physical portions of the CP/NET requester

At the user level, CP/NOS provides a virtual CP/M 2.X system to the requester microcomputer. A requester microcomputer can consist of no more than a processor, memory, and an interface to the network. Thus, a CRT with sufficient RAM can execute CP/M programs, performing its computing locally and depending on the network to provide all disk, printer, and other I/O facilities. Figure 1-2 illustrates the relationship between CP/NOS, MP/M II, and CP/NET.

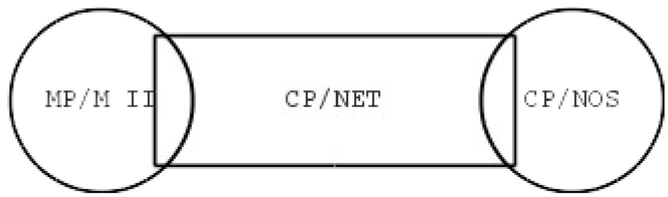


Figure 1-2. CP/NOS Configuration

1.1 CP/NET Features

CP/NET operates in multiple-processor environments ranging from tightly to loosely coupled to networked processors. In this manual, tightly coupled processors are those sharing at least a portion of common memory. Interprocessor messages communicate at memory speed. Loosely coupled processors do not have access to memory that is common or accessible by both processors; they communicate via a short, high-speed bus. Loosely coupled processors usually reside in the same physical box. Networked processors are usually physically separated and communicate over a serial link.

The CP/NET operating system is an upward-compatible version of CP/M 2.2, which provides system I/O facilities to requester microcomputers through a network. Additions to the Basic I/O System (BIOS) called the Slave Network I/O System (SNIOS), and a new Basic Disk Operating System (BDOS) called the Network Disk Operating System (NDOS), provide network access to System I/O facilities. The requester NDOS and NIOS are loaded and executed while running under CP/M 2.2.

In addition to the standard CP/M facilities, CP/NET provides the following capabilities:

- The network can be accessed for system I/O facilities.
- The network environment can be reconfigured to access I/O facilities according to application requirements.
- Messages can be transmitted and received between requesters and servers.
- An electronic mail system allows requesters and servers to send mail to each other.

The MP/M II server is implemented by adding some resident system processes at system generation (GENSYS) time. The resident system processes include server processes (SERVER) that perform the logical message-handling functions for the server and network interface processes (NETWRKIF) that you can customize for a particular hardware network interface.

1.2 CP/NET Configurations

CP/NET supports a number of different network topologies and a variety of system resources. The interprocessor message formats permit a requester to access more than one server for different resources.

Figure 1-3 illustrates an MP/M II system supporting a single CP/NET requester. The requester is a totally independent system, with its own console, printer, and disk resources. The requester can also access the MP/M II system's resources over the network. The MP/M II system also supports other users using local terminals.

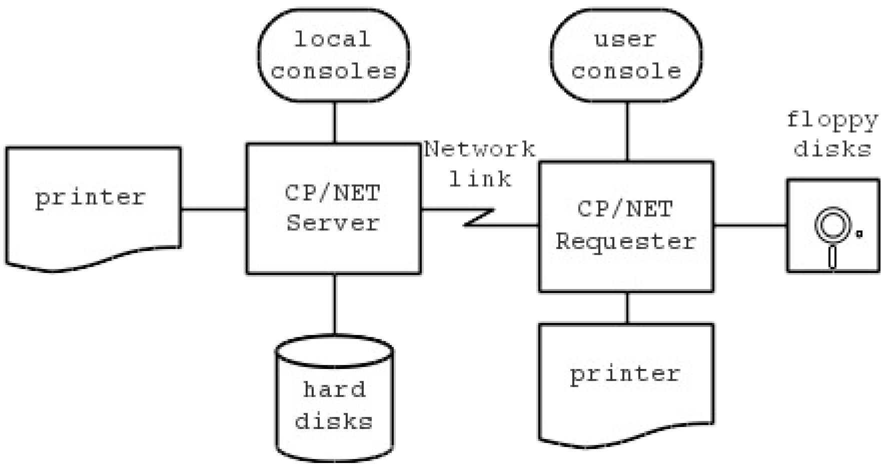


Figure 1-3. Single Requester Networked to MP/M II Server

Figure 1-4 shows an active hub-star network running CP/NET. Each requester is networked to the server through a unique network port. The requesters have their own local resources, but they also share the server's disk and printer resources. This topology is simple to implement because you can adapt the network protocol from the protocol used for RS-232 console drivers. The sample system in Appendix E uses this topology.

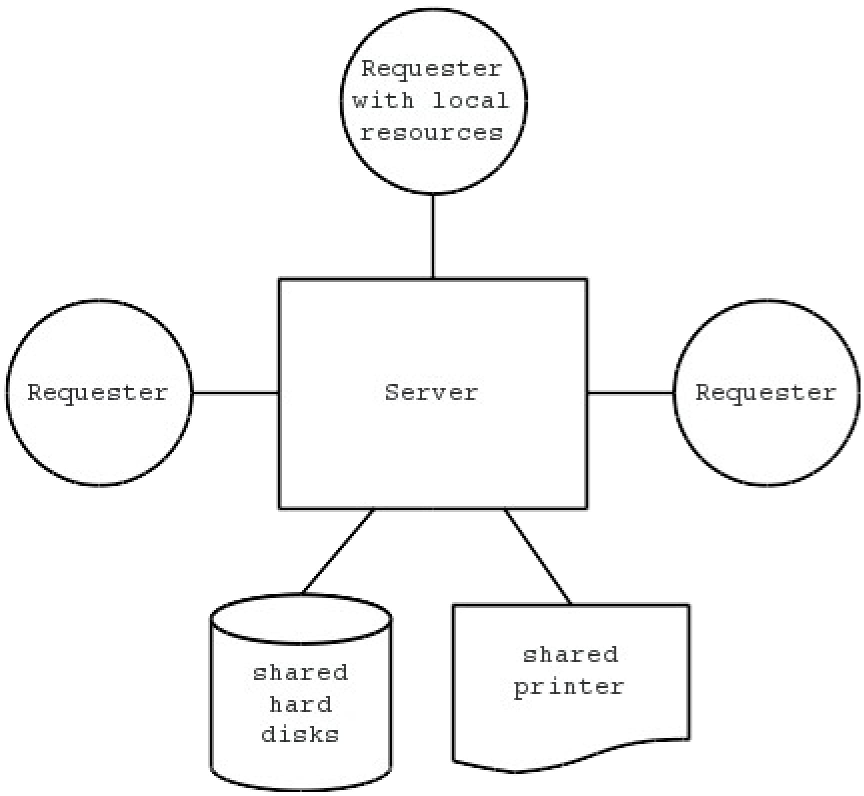


Figure 1-4. Multiple Requesters in Active Hub-star Configuration

Figure 1-5 shows a system of three requesters and two servers networked together in a bus or multi-drop configuration. The network protocol must be capable of resolving conflicts when nodes attempt to use the network simultaneously. Each requester has access to the resources of both servers, in addition to its own local resources. Appendixes F and G provide examples of CP/NET systems using this network topology.

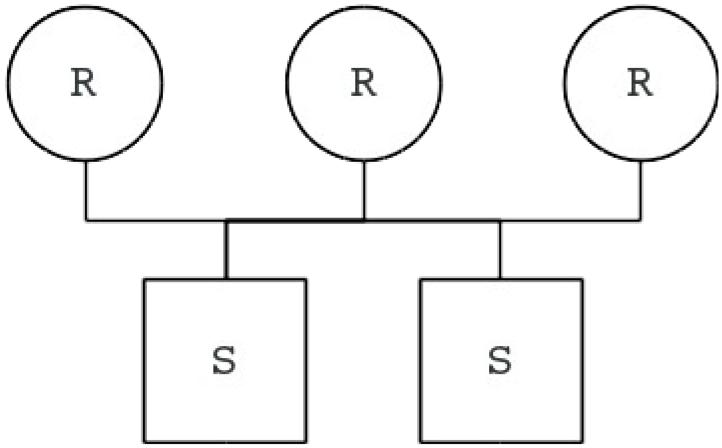


Figure 1-5. Multi-drop Network

Finally, you can combine these topologies, as well as other topologies like loops and trees, into a hybrid network topology. Figure 1-6 depicts such a topology, combining the bus, star, and loop forms.

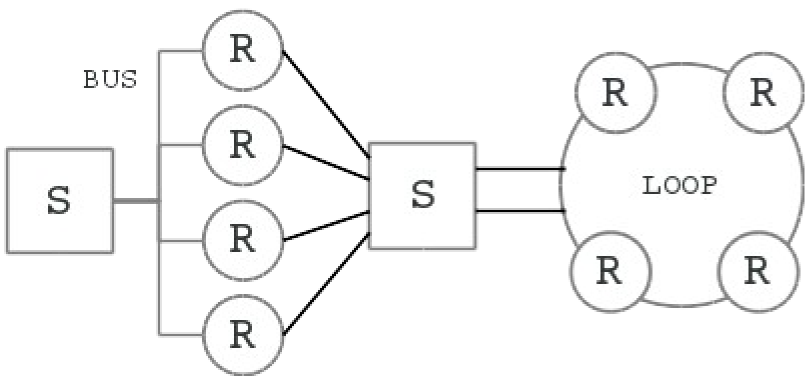


Figure 1-6. Hybrid Network

1.3 How the Requester Works

The CP/NET requester software runs under an unmodified CP/M version 2 operating system. The requester operating system consists Of three object modules: NDOS.SPR, SNIOS.SPR, and CCP.SPR. These modules are system page relocatable files that can be loaded directly under the CP/M BDOS and BIOS, regardless of their size or their location in memory.

The module NDOS.SPR contains the Network Disk Operating System (NDOS), the logical portion of the CP/NET system. The NDOS determines whether devices referenced by CP/M function calls are local to the requester or whether they are located on a remote system across a network. If a referenced device is networked, the NDOS, prepares messages to be sent across the network, controls their transmission, and finally reformats the result received from the network into a form usable by the calling application program. NDOS.SPR is distributed in object form by Digital Research. No modification to this module is required to run CP/NET.

The Slave Network I/O System (SNIOS) is contained in the module SNIOS.SPR. The systems implementer must customize this software to run on a particular computer and network system. The SNIOS performs primitive operations that allow the NDOS to send and receive messages across a network. The SNIOS also provides a number of housekeeping and status functions to the NDOS. Digital Research distributes a number of example SNIOS modules in source form with CP/NET.

The final module, CCP.SPR, is a replacement for the normal CP/M CCP. Like the regular CCP, CCP.SPR is loaded directly below the operating system. However, CCP.SPR performs a number of special network functions that initialize the environment for a program.

The logical origin of SPR files is location zero. Each file has a 256-byte header, with locations 1 and 2 defined as the length of the code in the file. A bit map, appended to the end of the code, identifies bytes of the code that must be relocated when the code is loaded on a particular page (256-byte) boundary.

The CP/NET utility CPNETLDR relocates the bytes defined by the bit map. CPNETLDR loads SNIOS.SPR directly below the CP/M BDOS. NDOS.SPR is loaded directly below the SNIOS. CPNETLDR then passes control to an initialization routine. This routine modifies key areas of the operating system:

1. Location 5, which contains a jump to the BDOS entry point, is saved away by the NDOS.
2. Location 5 is then modified to jump to an entry point in the NDOS. This assures that the NDOS intercepts all CP/M function calls.
3. The BIOS jump vector entries for console status, console in, console out, list status, list out, and warm boot are replaced with entries that jump into special NDOS routines. The NDOS saves the BIOS entry points for these routines, allowing direct BIOS calls to these routines to be intercepted in exactly the same way that CP/M function calls are intercepted.

After these modifications have been made, the NDOS calls the SNIOS to initialize the network. The NDOS then jumps to its own warm boot routine, which performs a disk system reset, loads CCP.SPR, and then passes control to the CCP.

When an application program calls the CP/NET operating system via location 5, the NDOS is entered instead of the BDOS. Invalid functions return to the user program immediately as errors. Functions dealing with console or printer I/O immediately pass through to the local BDOS; but these functions are intercepted by the NDOS again when the BDOS calls the BIOS. At this level, the NDOS checks whether the console or printer is a networked device. If so, the NDOS sends a request across the network for the input or output.

Some functions have no meaning when they are sent across the network to a remote server. Examples of these are Function 26 (Set DMA Address), Function 32 (Get/Set User Number), and Function 12 (Return Version Number). The local BDOS always handles these functions. But the NDOS saves certain parameters from these functions for its own use, processing them before allowing them through to the BDOS.

Finally, the NDOS checks most functions that deal with either the disk drive system or the file system to determine whether they reference local devices. If so, these functions pass unmodified to the BDOS. The NDOS also checks whether these functions reference devices that exist somewhere out on the network. If they do, the NDOS constructs a network message to be sent to the system on which the device exists. The network message contains the network function to be performed and the information necessary to perform it.

Figure 1-7 illustrates how the CP/NET operating system is organized. The solid line outlines the function flow of an operation on a networked disk drive. The dotted line traces the flow of an I/O operation to a networked list device or console. Arrows indicate possible function flow.

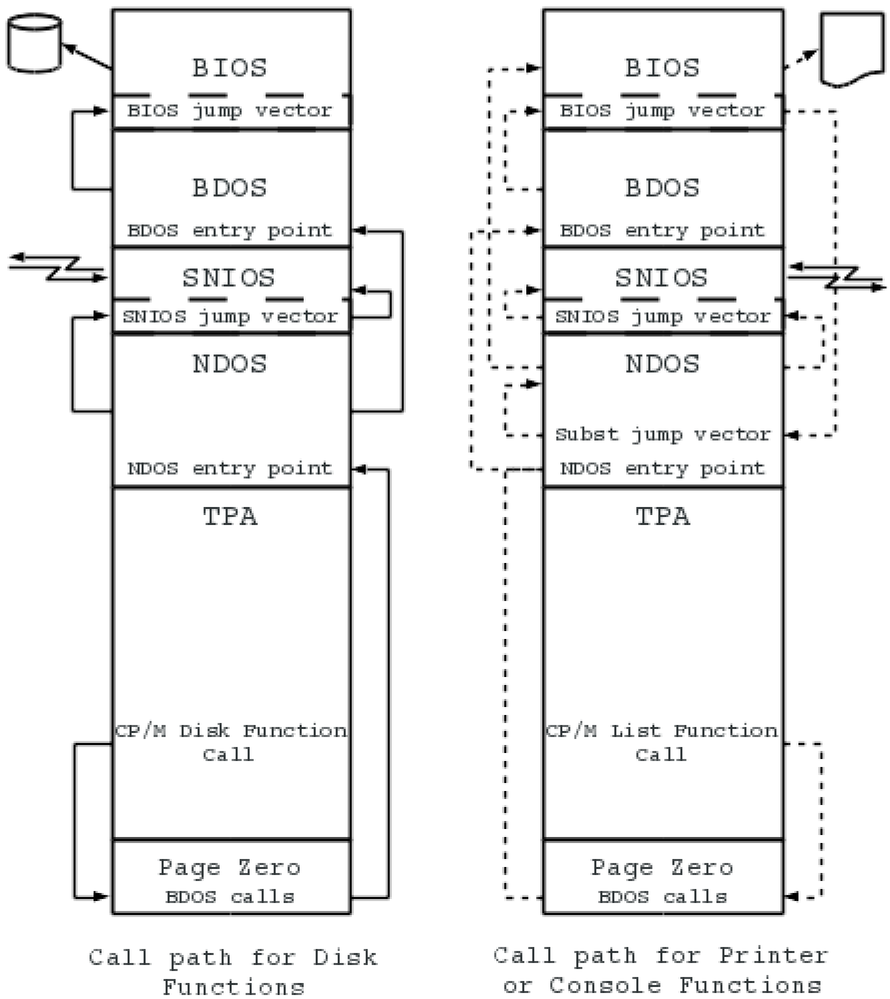


Figure 1-7. CP/NET Memory Structure

When an NDOS requester sends a function message out over the network, a response from the addressed server is implied. As soon as the NDOS has successfully called the SNIOS to send the message, the NDOS calls the corresponding message receive routine, also in the SNIOS. This procedure precludes the problem of trying to recover sequencing information from an arbitrary stream of messages.

The NDOS uses the network response to update the application program that made the function call. The NDOS then returns to the application program. If the device referenced was local, then the requester's BDOS updates the application program.

1.4 How the Server Works

Unlike the requester, the server software that runs under MP/M II does not modify the actual operating system. Rather, the operating system is a set of cooperating processes under MP/M II.

In its most basic form, each requester to be attached to a server requires two processes, communicating through two queues. One process, resident in the NETWRKIF.RSP module, performs the physical message transport task. The systems implementer must modify this process to accommodate the network's node-to-node protocol. The process's protocol must be compatible with that of the requester's SNIOS.

The NETWRKIF must be capable of monitoring one or more network lines in real-time and detecting when a requester is trying to send a message. The NETWRKIF must then receive the message, check it for data integrity, and send it on to the logical portion of the server, contained in the module SERVER.RSP. When the SERVER module returns its response to the logical message, the NETWRKIF must receive the message and then transmit it across the network back to the requester.

The module SERVER.RSP performs the logical operation the requester specifies. After receiving the message from the NETWRKIF, SERVER.RSP checks to make sure that the requester is logged in properly. Then SERVER.RSP responds to the message by performing a series of MP/M II operating system calls. Using the information returned by those calls, the SERVER constructs a response message and sends it to the NETWRKIF module for transmission.

Both the NETWRKIF and SERVER modules are Resident System Process files (RSPs). RSPs are built into the MP/M II system during its GENSYS operation. When MP/M II is cold started, all RSPs are automatically dispatched. Each RSP module might contain multiple processes, but only one process per RSP is automatically dispatched. Because each requester bound to a server might require one process from the NETWRKIF and one from the SERVER, both RSPs contain initialization code to create additional copies of themselves. These processes can be reentrant. They can share the same code, but they have separate data areas to avoid conflict between program variables.

One of the simplest server architectures is shown in Figure 1-8. Processes from the NETWRKIF are named `NtwrkIP<x>` where `<x>` is the ASCII representation of a hexadecimal number between 0 and F. SERVER processes are named `SERVR<x>PR`.

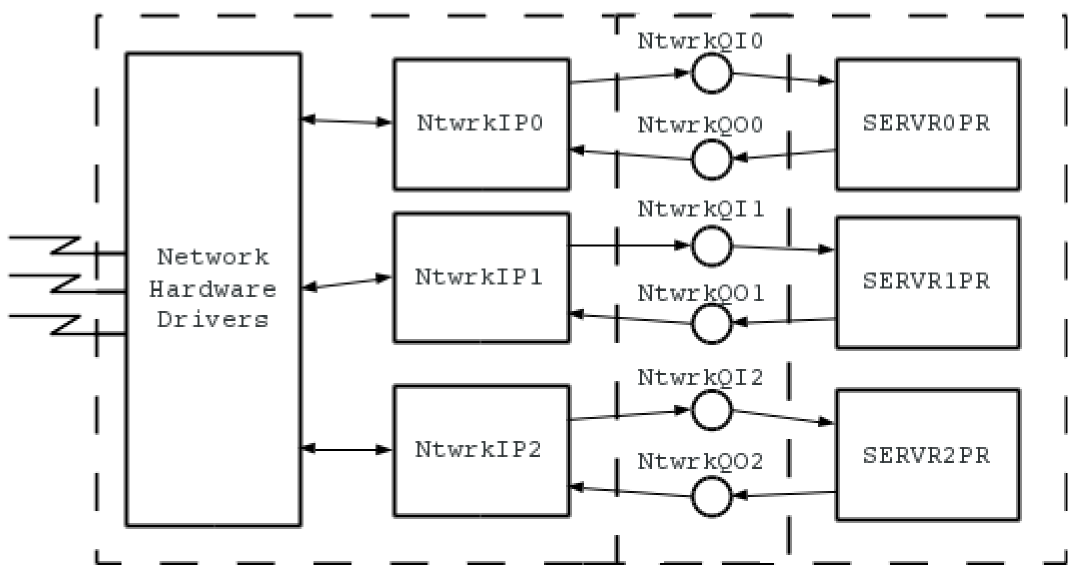


Figure 1-8. A Simple Server that Supports Three Requesters

A NtwrkIP<x> process writes the address of an input message to a queue named NtwrkQI<x>. A SERVR<x>PR process reads this queue while waiting for an input message. Because the queue is empty when the requester is not requesting service, the SERVR<x>PR process is suspended and consumes no CPU resources.

When the NtwrkIP<x> process writes to the queue, the SERVR<x>PR process is dispatched, and it begins to operate on the message. As soon as the NtwrkIP<x> process has finished sending the incoming message to NtwrkQI<x>, NTWRKIP<x> immediately tries to read a second queue, named NtwrkQO<x>. This queue is empty, and the NtwrkIP<x> process is consequently suspended until the SERVR<x>PR process writes the response message to it. The NtwrkIP<x> can then transmit the message back to the requester.

Server functions can be divided into four categories:

- session control functions
- file serving functions
- print serving functions
- non-CP/NET functions

Session control functions permit a requester to log on to a server, log off, set compatibility attributes, set default passwords, and examine the server configuration table.

File serving functions make up the bulk of the server's work. These functions include opening and closing networked files, reading and writing files, and managing disk devices.

The server can operate as a print server in two different modes. If the MP/M module SPOOL.RSP is present in the system, requester outputs to a networked list device are spooled to a file for future printing. If no spooler exists in the system, the server manages the attaching and detaching of various print devices.

Finally, the NETWRKIF module can be designed to recognize a logical message that has no meaning to the SERVER module, but that can be operated on by a user-defined process. This feature allows you to use functions CP/NET does not provide.

Section 2
CP/NET User's Guide

This section describes the requester commands that enable you to access the network and use its resources. All the requester commands are actually COM files that reside on disk at the requester.

2.1 The LOGIN Command

The LOGIN command allows a requester to log in to a specified server. A requester must log in before any resources on the server can be accessed. Once a requester has logged in, it is not necessary to log in again even though the requester might power down and then power up again. A requester can only be logged off a server by an explicit LOGOFF command issued from the requester. The command takes the general form:

```
LOGIN {password}{[mstrID]}
```

where `password` is an optional 8 ASCII-character password; the default password is PASSWORD. `[mstrID]` is an optional two-digit server processor ID; the default is [00]. The simplest form is

```
A>LOGIN
```

2.2 The LOGOFF Command

The LOGOFF command allows a requester to log off from a specified server. Once a requester has logged off, the server cannot be accessed again until you issue a LOGIN command. The command takes the general form:

```
LOGOFF {[mstrID]}
```

where `[mstrID]` is an optional two-digit server processor ID; the default is [00]. The most simple form is

```
A>LOGOFF
```

2.3 The NETWORK Command

The NETWORK command enables a requester to assign selected I/O to the network. The NETWORK command updates the requester Configuration table. The command takes the general form:

```
NETWORK {local dev}{=}{server dev{[srvrID]}}
```

where `local dev`server dev is the specification of a server device such as A:, B: ... P: in the case of a disk device or 0, 1 ... 15 in the case of CON: or LST:. A missing `server dev` defaults to 0 in the case of CON: or LST:. `[srvrID]` is an optional two-digit hexadecimal server processor ID. The default is [00]. Typical assignments are

```
A>NETWORK LST:

A>NETWORK LST:=3[07]     (list dev #3 on server 07)
A>NETWORK CON:=2         (console #2 on dflt srvr)
A>NETWORK B:=D:[F]       (logical B: is D: on server 0F)
```

Note: when networking drive A: to a server, the file CCP.SPR must reside on the networked drive, or warm boot operations fail. Do not network a device to a nonexistent or off-line server because network errors could result.

2.4 The LOCAL Command

The LOCAL command enables a requester to reassign selected I/O back to local from the network. The LOCAL command updates the requester configuration table. The command takes the general form:

```
LOCAL {local dev}
```

where `local dev` is the specification of a local device such as LST:, A:,... CON:. The following are typical assignments:

```
A>LOCAL LST:
A>LOCAL B:
```

2.5 The ENDLIST Command

The ENDLIST command sends a hexadecimal 0FF to the list device, signaling that a list output to a networked printer is finished. If a spooler is resident on the server, the spool file is closed and enqueued for printing. If no spool file is present, the networked list device is freed for use by another requester.

Note: the CCP implements an endlist every time a program terminates, provided that CTRL-P is not active at the time. Turning CTRL-P off also causes an endlist.

```
A>ENDLIST
```

2.6 The DSKRESET Command

The DSKRESET command functions exactly like the PRL that executes under MP/M II. DSKRESET resets the specified drive, so a disk can be changed. The command takes the general form:

```
DSKRESET {drive(s)}
```

where `drive` is a list of the drive names to be reset. If any of the drives specified cannot be reset, the console displays the message:

```
***Reset Failed***
```

The following are typical disk resets:

```
A>DSKRESET          (resets all drives)
A>DSKRESET B:,F:    (reset drive B: and F:)
```

2.7 The CPNETLDR Command

The CPNETLDR command loads the requester CP/NET system. Specifically, the SNIOS.SPR file loads and relocates directly below the CP/M BDOS. The NDOS. SPR file loads and relocates directly below the SNIOS.

From that point on, the BIOS, BDOS, SNIOS, and NDOS remain resident in memory. The CPNETLDR requires no user customization. CPNETLDR displays an error message when loader errors are encountered. Listing 2-1 is a typical CPNETLDR execution.

```
A>CPNETLDR
CP/NET 1.2 Loader
---------------
```

```
BIOS          F600H 0A00H
BDOS          E800H 0E00H
SNIOS   SPR  E500H 0300H
NDOS    SPR  DB00H 0A00H
TPA          0000H DB00H

CP/NET 1.2 loading complete.
<Warm Boot>
A>
```

Listing 2-1. A Typical CPNETLDR Execution

2.8 The CPNETSTS Command

The CPNETSTS command displays the requester configuration table. The requester configuration table indicates the status of each logical device that is either local or assigned to a specific server on the network. Listing 2-2 shows a typical CPNETSTS execution.

```
A>cpnetsts

CP/NET 1.2 Status

Requester processor ID = 34H
Network Status Byte = 10H
Disk device status:
   Drive A: = LOCAL
   Drive B: = LOCAL
   Drive C: = Drive A: on Network Server ID = 00H
   Drive D: = Drive B: on Network Server ID = 00H
   Drive E: = LOCAL
   Drive F: = LOCAL
   Drive G: = LOCAL
   Drive H: = LOCAL
   Drive I: = LOCAL
   Drive J: = LOCAL
   Drive K: = LOCAL
   Drive L: = LOCAL
   Drive M: = LOCAL
   Drive N: = LOCAL
   Drive O: = LOCAL
   Drive P: = LOCAL
Console Device = LOCAL
List Device = List #0 on Network Server ID 00H
A>
```

Listing 2-2. A Typical CPNETSTS Execution

2.9 CTRL-P

A CTRL-P causes console output to be echoed to the list device until the next CTRL-P. The messages

```
CTL-P ON
```

and

```
CTL-P OFF
```

are displayed at the console. When the requester list device has been networked, the local system uses the server printer. The second CTRL-P causes a hexadedimal FF to be sent to the server, causing the server to close and print the spool file.

Note: when the requester uses the server printer with a CTRL-P active, the requester must issue a second CTRL-P to cause the server to close the spooled file and begin printing it. When the requester is using the server printer and has invoked it with a program such as PIP, the warm boot at program termination causes the required endlist character to be sent to the server to close and print the spooled file.

The program ENDLIST is not needed to terminate network list output in these situations.

2.10 The MAIL Utility

The MAIL utility allows you to send, receive, and manage electronic mail in a network environment. MAIL operates using file based function calls, so special processing by the server is not required. MAIL runs transparently on either server or requester, so only one program is required throughout the entire electronic mail system.

MAIL allows you to send messages to a single node, broadcast messages to all nodes currently logged in, or receive messages.

Messages are stored for your future examination on the temporary file drives of CP/NET servers. A user's mail file is named

```
xxMAIL.TEX
```

where xx corresponds to your node ID. For example, if requester #5C wants his mail, the MAIL program accesses files named 5CMAIL.TEX on the temporary file drives of all the servers that node 5C currently has logged in. Every server in the CP/NET system might have one of these files, so other nodes in the network that do not have direct access to all of node 5C's servers can still send messages indirectly to it.

Menu-driven operation allows you to run the program with a minimum of instruction. Messages are limited in size to 1.7K bytes. You can enter messages into the system directly from the keyboard or through a preedited file. Options allow you to answer a message immediately while reading your mail and to delete unwanted entries.

2.10.1 Menus

Three basic menus can appear during a MAIL session:

- Main Menu
- Input Source Menu
- Receive Response Menu

The Main Menu determines the basic operation to be performed. The Input Source Menu specifies whether input comes from a file or whether you enter it directly. Finally, the Receive Response Menu determines the disposition of messages you receive.

Enter a menu selection by typing the number associated with the selection, followed by a carriage return. If you type an invalid character or no character at all, the menu system defaults to the last item on the menu. You simply press the carriage return for common operations.

Main Mail Menu

The main mail menu appears when you enter the mail program and when any of its options have completed execution. Main mail menu options are

```
1 - Broadcast
2 - Send Mail
3 - Receive Mail
4 - Exit Program
```

A simple carriage return or an invalid entry at this level return you to CP/M or MP/M II command level.

Input Source Menu

The input source menu allows you to specify how message input is entered into the system. The input source menu has only two options:

```
1 - File
2 - Console Input
```

Receive Response Menu

The receive response menu determines the disposition of messages once the user has examined them. The options are

```
1 - Stop Receiving Mail
2 - Answer Message
3 - Delete Message From Mail File
4 - Answer Message, Then Delete
5 - Re-Examine Last Message
6 - Get Next Message
```

2.10.2 Data Entry

In addition to the menus, MAIL prompts you for a variety of inputs. These inputs determine the destination of messages, input files, and subjects.

Destination ID Prompt

When using the send mail option, MAIL requires an explicit destination to deliver the message properly. The system prompts for the destination. The legal value is a 2-digit hexadecimal number, followed by a carriage return. This value corresponds to a CP/NET server or requester ID value.

If you enter a value that is not a legal hexadecimal number, the system displays an error message, and prompts you again. The system does not check, however, to determine whether a requester or server with this ID exists on the network.

Subject Prompt

With both the broadcast and send mail options, MAIL prompts for a subject header. This header is displayed as the title of the message and is also used for answering mail to the message that is sent.

When the system prompts for subject, you can enter a subject header from 0 to 80 bytes long, followed by a carriage return.

Input File Prompt

If a preedited file contains the text of a message, MAIL prompts for the filename. You can then enter a valid CP/M file specification. If the file specified does not exist, the system displays an OPEN ERROR, and the program aborts.

Console Input Prompt

If you choose to enter a message directly from the console, MAIL prompts for input. You can then simply type the message. Individual message lines can be up to 78 characters long. A message, whether input from the console or from a file, must be no longer than 1764 characters, about enough to fill a standard terminal display. Longer messages are truncated.

To terminate input, the user presses CTRL-Z, followed by a carriage return.

2.10.3 MAIL Options

This section explains how the CP/NET system gathers and receives mail and how you control the disposition of mail.

Broadcast

The broadcast option sends a message to every node that it can find logged in to the CP/NET system.

MAIL works differently when it is running on a server under MP/M II, from the way it works when it is running on a requester under CP/M or CP/NOS. If a requester is broadcasting, MAIL sends the specified message to every server on which it is logged in as well as to every other requester logged in to those servers. If a server is broadcasting, MAIL sends the message only to every requester logged in to that server. A server has no means of initiating transactions with other servers, although it can use its own local MP/M II system to file mail for its own requesters.

A message cannot be broadcast to the broadcasting node.

To send a message to a given server and its associated requesters, MAIL must reference that server's temporary file drive across the network. If a requester has not networked the temporary file drive of a server, no messages are sent to that server.

When the broadcast option is entered, MAIL prompts you for a subject and message. When the operation is completed, it returns to the main menu.

Send Mail

The send mail option sends a message to a specific node in the CP/NET system. The destination can be either a server or a requester. If the option is running on a requester, it first searches the network to see if the node specified is logged in. If the option finds the node is logged in, it sends the message. if the option does not find the node, it leaves the message on the first server located when MAIL searches the local configuration table. If a destination requester logs in later, its mail will be waiting for it. Mail files can accumulate that were erroneously sent to nonexistent requesters or to servers that the requester sending the message had not logged onto when it sent the message.

If the option is running on a server, mail is left on that server, whether the node it is being sent to is logged in or not.

Upon selecting the send mail option, MAIL prompts you for a destination ID, a subject, and for the message itself. MAIL then attempts to send the message. If MAIL cannot find a server with a temporary file drive to accept the message, the error NO SERVER MAIL DRIVE NETWORKED is displayed, and the program aborts.

Receive Mail

The receive mail option permits you to examine messages left for you on all the servers on which you are currently logged in. After each message is displayed, you are presented with a number of message-handling options.

If you are running MAIL on the server, only the mail file on the server is accessed. However, if MAIL is being run on a requester, each server to which the requester is logged in is searched for messages.

Each message is preceded by a header that tells you what node the message came from and the subject of the message. The actual message is then displayed. As a message is being displayed, you can halt the display by pressing CTRL-S and resume display by pressing CTRL-Q. At the end of the message, bring up the receive response menu by pressing any key. You can then take one of the options listed in Table 2-1.

Option	Explanation
Stop receiving mail	MAIL stops searching for more entries or additional files and returns to the main menu.
Answer message	MAIL prompts you to type in a reply message. The reply message is sent back to the sender of the original message. The subject of the reply message is the characters "RE: ", followed by the original subject.
Delete message	MAIL flags the message in the file as deleted. At the end of each file, or if you decide to stop receiving mail, deleted messages are physically removed from the file.
Answer, then delete	This option answers the message message just displayed, then deletes the message.
Display next message	Messages continue to be displayed in this fashion, allowing the user to respond to each one, until no more can be found. The message "No More Messages" is then displayed, and the program returns to the main menu.

Table 2-1. Receive Mail Message-handling Options

Upon completion of any message-handling options, with the exception of the reexamine option, the next message is displayed.

2.10.4 Error Messages

In addition to the error messages already mentioned, CP/NET returns file system errors. These errors display

```
ERROR READING FILE
ERROR WRITING FILE
```

```
or
ERROR OPENING FILE
```

followed by a filename. After displaying such an error, MAIL aborts.

It is possible to get the ERROR OPENING FILE message by specifying a nonexistent input file for sending or broadcasting a message. Almost all other instances of the messages, however, indicate possibly serious trouble with the network, the server file system, or the mail-handling system.

Section 3
CP/NET Programmer's Guide

This section provides information for the applications programmer who wants to write programs to run under CP/NET or to evaluate the performance and correctness of programs written for CP/M or MP/M II under the CP/NET operating system.

MP/M II performs all operations on a networked device and makes file security checks that CP/M does not usually make. Because MP/M was designed to run unmodified CP/M applications, these checks seldom prevent the use of a CP/M application under CP/NET.

3.1 CP/NET Interprocessor Message Format

The simple message format that CP/NET uses for interprocessor communication includes packaging overhead and the message itself. The packaging overhead is a header consisting of a message format code, a CP/NET destination address, a CP/NET source address, a CP/M function code, and a message size. The actual CP/NET message follows the header.

3.1.1 Message Format Code

The message format code is a single byte that specifies the format of the message itself. Digital Research reserves message formats 0-127 for general interprocessor message format codes and future use. The general interprocessor format codes follow the message format shown below, but differ in length of the individual fields. (See Appendix B.)

The odd-numbered format codes are for response messages sent back from servers to requesters. Thus, a CP/M disk read function sent from a requester to a server has a message format code of 0, and the return code sent back from the server to the requester has a message format code of 1.

Implement the general interprocessor message formats 0 and 1 as shown in Appendix A because these formats promote standardization among microcomputers from different vendors.

3.1.2 Message Destination Processor ID

The message destination processor ID field is one byte long. Destination IDs can be in the range O-0FE hex. An ID of 0FF is illegal. Many CP/NET utilities use a server destination of 0 as a default. For this reason, assign the most commonly used network server a node ID of 0.

3.1.3 Message Source Processor ID

The message source processor ID field is usually one byte long. The node sending the message always fills this field with its own ID. Valid source IDs range from 0 to 0FE hex. An ID of 0FF is illegal.

3.1.4 CP/M Function Code

The CP/M function code field is one byte long. The size of the message data field depends on the CP/M function. Each CP/M function has a specific number of bytes to be sent to the server and a specific number of bytes to be returned to the requester. Appendix C provides the logical message specification for each of the CP/M functions. Some of the CP/M function codes have no equivalent network function.

3.1.5 Size

The size field is one byte long. The size value has a bias of 1. Thus, a size of 0 specifies an actual size of 1, while a size of 255 specifies an actual size of 256. With a 1-byte size field, the minimum data field is 1 byte, and the maximum is 256.

3.1.6 CP/NET Message

The CP/NET message consists of binary data and is from 0 to 256 bytes long. The meaning of the message depends on the format, function, and size specified by the header.

3.1.7 Additional Packaging Overhead

Some networks might have to modify the standard CP/NET message to transmit it over the physical network medium, route it to the proper destination, and ensure its integrity.

For example, the message format shown in Figure 3-1 contains no cyclic redundancy code (CRC) or any other error checking as a part of the packaging overhead. The user-written SNIOS can add the error checking when it places the message onto the network, and then test the message when the SNIOS receives a message from the network. This function is intentionally left to the user, avoiding redundant error checking where standard interface protocols, both in software and hardware, might already provide error checking.

The NDOS always constructs messages using format 0. Likewise, the server processes always expect to receive messages in format 0. The server sends its response in format 1, which the NDOS requires to interpret the response. If the SNIOS and NETWRKIF must communicate using a different format, they must convert all received messages back into the standard formats 0 and 1.

```
+-----+-----+-----+-----+-----+-----      ------+
| FMT | DID | SID | FNC | SIZ | MSG  ...        |
+-----+-----+-----+-----+-----+-----      ------+
   ^     ^     ^     ^     ^      ^
   |     |     |     |     |      |
   |     |     |     |     |      +-- Message of length SIZ+1 bytes
   |     |     |     |     +-- Size of message = message length - 1
   |     |     |     +-- CP/NET Function Code
   |     |     +-- Message Source ID
   |     +-- Message Destination ID
   +-- Format Code
```

Figure 3-1. Message Format

3.2 Running Applications Transparently under CP/NET

Applications that use local devices under CP/NET use the CP/M 2.2 BDOS file system. Applications that use networked devices use he MP/M II file system. These operating systems are largely compatible with each other, so applications written to run under CP/M should run across the network with no changes.

But there are some differences between the two file systems:

- The CP/NET NDOS supports MP/M II functions not supported under CP/M 2.2. Because these function calls are meaningless to CP/M, they can only be made to devices that are mapped across the network.
- The two operating systems handle errors differently. The NDOS reconciles these differences, for CP/M application programs. A special function call takes advantage of MP/M II's extended error-handling capability for applications referencing networked devices under CP/NET.

- MP/M II file security checking can cause certain CP/M applications to abort because these applications modify fields in the File Control Block that make the FCB invalid to MP/M II. Special compatibility modes have been added to CP/NET to allow these applications to run without modification.
- Temporary filenames, like $$$.SUB or FILENAME.$$$, are modified under CP/NET. If more than one requester requires a temporary file with the same name, this modification prevents collisions between filenames that otherwise cause an application to abort. The modification is transparent to the application, but it can be confusing when trying to analyze aborted programs.
- A CP/NET requester presents a different version number to an application program when it calls Function 12 (Return Version Number). Under CP/M 2.2, this function returns a 002x value. Under CP/NET, it returns a 022x value. Application programs checking this version number might not function properly. They must be modified. Modifications to CP/NET, to present the same version number as CP/M, are now included as application notes in all releases of the CP/NET product.
- You can protect files on networked drives from unauthorized access by requiring a requester to specify a predefined password. You can also assign default passwords to all servers logged on to a particular requester.
- Certain files that exist only on user zero can be opened by any other user number if they are opened in the proper mode.
- The operating system must handle the printer differently under CP/NET from under CP/M because printer output is buffered into 128-byte packets. The operating system must have some way of deciding when an application program has finished using the printer. Also, several requesters might be competing for the same printer.
- The allocation vector for a networked drive is returned into the NDOS's default message buffer on a call to function 27 (Get Allocation Vector Address) and register pair HL is set to the address of the message buffer. Because of this, the allocation vector must be used or moved before the next network message is sent, or the vector is destroyed.

Differences between the CP/M 2.2 BDOS and MP/M II file systems are more fully described in the following sections.

3.2.1 MP/M II vs. CP/M File Systems

MP/M II is a real-time, multitasking operating system. To function properly, MP/M II requires a file system capable of sharing files among multiple processes and resolving access conflicts among those processes. In contrast, CP/M is a single-task operating system, so no such conflicts can arise.

One of MP/M II's key methods for maintaining file system integrity is the File Control Block checksum. The FCB checksum takes into account the process controlling the FCB, the physical blocks allocated to the file, whether the file is open in a mode that allows other processes to share it, and other factors. When file-related functions are submitted to MP/M II, the checksum is examined. If the checksum is found to be invalid, MP/M II returns an error to the calling process.

MP/M II also returns an error if

- a process attempts to open a file in a mode incompatible with the mode of a file already opened by another process
- a valid password is not supplied for the file
- a user tries to write to a file opened for Read-Only access
- a process exceeds certain predefined parameters for the operating system

Because a single process handles all CP/NET activity on a server all of these limitations apply to a CP/NET requester performing file operations on a remote device. These limitations, however, do not apply to a requester accessing a local device. The systems implementer should take these factors into account when designing servers for a CP/NET system.

3.2.2 Error Handling Under CP/NET

Most CP/NET function calls result in specific values returned in the CPU registers. These values can be pointers to data objects, bit vectors specifying drive status, directory codes, or success or error conditions. Directory, success, and error codes are returned in register A. Pointers and bit vectors are returned in register HL. Register A is always equal to register L, and register B is equal to register H for all CP/NET return codes.

Error Handling for Local Devices

When a CP/NET requester performs a local file operation, the function parameters pass untouched to the CP/M BDOS. The BDOS checks those parameters for validity and calls the BIOS to perform physical I/O functions. Two types of errors can arise from these local operations.

The BDOS can detect certain logical problems with a file function and return a logical error. If it does, an error code is returned in register A, but the calling application program is allowed to continue.

A physical error is returned when the BIOS is unable to successfully perform a physical operation requested by the BDOS. When the BDOS is presented with a physical error, it prints the following message on the console:

```
BDOS Err on <x>:
<error message>
```

where <x> is the drive referenced when the error occurred, and <error message> is one of the four following errors:

- Bad Sector
- Select
- File R/O
- R/O

After the physical error message is printed, the BDOS waits for the user to respond to the error with one of two actions. Pressing CTRL-C causes the BDOS to perform a warm boot, aborting the program. Pressing any other key causes the BDOS to ignore the physical error and continue as if it had not occurred.

For a more complete discussion of CP/M 2.x errors, see the CP/M Operating System Manual, published by Digital Research.

Error Handling for Network Devices

When an application references a networked device, the MP/M II server performs the actual file operation and returns a message defining whether the operation was successful or not. Unlike the local case, the requester has only indirect knowledge of any error status. Direct physical error indications are impossible to obtain because a requester has no contact with the MP/M II XIOS. Instead, if an error occurs, MP/M II returns a message indicating that an error occurred and the type of error it was.

When referencing a remote device, the two types of errors possible under CP/NET are logical errors and extended errors.

Like logical errors under local CP/M, logical network errors define nonfatal error conditions, such as reading past the end of a file or attempting to open a nonexistent file. Some serious error conditions are returned as logical errors for functions that expect to process their own errors. These functions are

20	Read Sequential
21	Write Sequential
33	Read Random
34	Write Random
40	Write Random with Zero Fill
42	Lock Record
43	Unlock Record

Errors for these functions are returned in the return code field of a CP/NET message. The NDOS formats this field into register A, so the condition code upon return to the application program looks exactly as it does under local CP/M.

Some of the following codes can be returned in register A for each of the preceding functions:

00	Function Successful
01	Reading Unwritten Data or No Directory Space Available
02	No Available Data Block (Disk Full)
03	Cannot Close Current Extent
04	Seek to Unwritten Extent
05	No Directory Space Available
06	Random Record Greater than 3FFFF
08	Record Locked by Another Process
09	Invalid FCB
0A	FCB Checksum Error
0B	File Verify Error
0C	Record Lock Limit Exceeded
0D	Invalid File ID
0E	No Room in System Lock List

Extended errors indicate that a potentially fatal condition has occurred during the execution of an MP/M II function. The condition can be a physical error, similar to the physical errors that can occur under CP/M. Or the condition can be an error produced by the file system, indicating that the specified operation violates the integrity of the file system.

When an extended error occurs under MP/M II, the default mode of operation displays the extended error message on the console attached to the calling process, and the process aborts, MP/M II provides, however, for returning extended errors to the calling process without aborting that process. In this return error mode, register A is set to FF hexadecimal, and register H contains the extended error code.

The CP/NET server uses return error mode because if the server aborted, it could not communicate further with the requester it was servicing until MP/M II was restarted. When the server detects an extended error, it constructs a special CP/NET message. The message is two bytes long, with the first byte (the return code) set to FF. The second byte is set to the extended error code.

When the requester detects one of these special messages, it checks the error mode set by the application program with Function 45 (Set BDOS Error Mode). There are three possible modes:

- Default Mode
- Return Error Mode
- Return and Display Error Mode

If the NDOS is in default mode, it prints the following error message:

```
NDOS Err <xx>, Func <yy>
```

where <xx> is the extended error code in hexadecimal, and <yy> is the function being performed when the error occurred, also in hexadecimal. The NDOS then performs a warm boot, aborting the program.

In return error mode, the NDOS does not display a message or abort the program. Instead, the NDOS sets register A to FF and register H to the extended error code; then it returns to the application program.

If an extended error is detected in return and display error mode, the NDOS displays the error message on the console. But the NDOS does not abort the program, setting the registers in the same manner as return error mode.

Function 45 (Set BDOS Error Mode) does not exist under CP/M. Because of this, most CP/M applications automatically run in default mode. If an extended error occurs, these applications abort.

The following extended error codes can be returned to the NDOS:

01	Bad Sector--Permanent Disk Error
02	Read-Only Disk
03	Read-Only File
04	Drive Select Error
05	File Open by Another Process in Locked Mode
06	Close Checksum Error
07	Password Error
08	File Already Exists
09	Illegal ? in an FCB
0A	Open File Limit Exceeded
0B	No Room in System Lock List
0C	Requester not Logged on to Server or Function Not Implemented on Server
FF	Unspecified Physical Error

Extended error 0C hex is returned, not by MP/M II, but by the server itself. This error indicates that the server is unable to process an otherwise valid CP/NET message, either because the requester is not logged in to that server or because the function code contained in the message is invalid.

Extended error FF can result only from two special functions, Get Allocation Vector Address and Get Disk Parameter Address. Because these functions return a pointer in register pair HL, it is not possible to detect a regular extended error. Instead, these functions return an FFFF value in HL if a physical error occurs. The

NDOS ensures that the address returned for these functions (including Get Server Configuration) never return an address with FF in the low byte, so if they return with A (or L) = 0FFH then the caller should assume an error.

Not all CP/NET functions are capable of returning extended errors. However, extended error 0C can be returned on any function, even on MP/M II functions that normally have no extended error associated with them. If an extended error is returned for such a function, the NDOS ignores it. The following functions can result in the performance of a network access but cannot produce an extended error:

1	Console Input
2	Console Output
5	List Output
9	Print String
10	Read Console Buffer
24	Return Login Vector
28	Write Protect Disk
29	Get Read-Only Vector
37	Reset Drive
39	Free Drive
64	Login
66	Send Message on Network
67	Receive Message on Network
70	Set Compatibility Attributes
106	Set Default Password

Any other function can cause a program to abort if an MP/M II extended error occurs, if an unsupported function is passed to the server, or if the server is not logged in.

3.2.3 Temporary Filename Translation

Many common application programs use temporary files. The names of these files often have the form FILENAME.$$$ or $$$.SUB. When multiple copies of these applications run on different requesters logged on to the same server, a number of these temporary files can have the same name, causing extended MP/M II errors that abort the application program.

To solve this problem, each requester's NDOS recognizes temporary filenames destined for networked drives and implicitly renames them, so the filename an application presents to the operating system is not the one the NDOS presents to the MP/M II file system.

Each occurrence of the string $$$ in the first three bytes of a filename, as well as any filetype of $$$, forms a CP/NET message with a filename or filetype of $<xx>, where <xx> is the ASCII representation of the requester ID byte. Because all requesters have a unique ID, this modification guarantees the uniqueness of temporary filenames.

This modification is transparent to the calling application program. When the NDOS modifies a filename in a CP/NET message, it converts the filename back to its original form before updating the application's FCB. The only possible change to the FCB is that interface attributes set in the high-order bits of the filename strings modified are reset. This change poses no problems if temporary files are truly temporary. Treat temporary files like Read-Write files with the DIR attribute; delete them before the application program terminates.

Functions 17 (Search For First Directory Entry) and 18 (Search For Next Directory Entry) do not perform temporary filename translation when referencing a networked drive. If a user creates file with a temporary filename and then attempts to locate it within his directory, this can be confusing.

For example, suppose that a user working on requester 5A enters the command:

```
REN $$$.$$$=BLAH.TMP
```

Then the user enters a DIR command. The file previously renamed appears as

```
$5A.$5A
```

in the directory.

If a temporary file is referenced on a drive that is local to the CP/NET system, the filename passes unmodified to the BDOS. -No conversion is necessary, because there is no possibility of conflict.

3.2.4 Opening System Files on User 0

Under MP/M II, a requester running in a user number other than 0 can access certain networked files in user 0. If an MP/M II file has its t2' interface attribute set, the file is a system file. If a networked file is opened in locked or Read-Only mode from a nonzero user number, the following actions are taken:

- If the file exists in the same user number, MP/M II opens the file.
- If the file does not exist in the same user number, MP/M II searches user 0. If the file exists on user Of and it is a system file, MP/M II opens it just as though the file existed under the other user number.
- If the file exists on user zero as a system file, but it is also a Read-Only file (interface attribute t1'), MP/M II automatically opens the file in Read-Only mode.

The user of a CP/NET requester can make convenient use of these options. Because the CCP.SPR always opens files in Read-Only mode, all COM files can be placed in user 0 and marked as system files, making them accessible to all user numbers.

Because this facility does not exist under CP/M 2.x, all COM files on local devices must exist within the user numbers from which they are to be executed.

3.2.5 Compatibility Attributes

Because of MP/M II's added file security, applications written under CP/M might not work properly under MP/M II. Two basic factors contribute to the incompatibility. The first is the FCB checksum computation that MP/M II performs on open FCBs. Certain CP/M applications modify their FCBs in a way that makes their checksums invalid. Second, MP/M II defaults to opening all files in locked mode, allowing only one process to have a file open at a time. Although files can be opened in an unlocked or shared mode, an application must explicitly specify that the file is to be opened unlocked. CP/M applications have no knowledge of this procedure.

To enable CP/M applications to run unmodified under MP/M II, a system of compatibility attributes has been added. This feature is supported under CP/NET. Using compatibility attributes, a user can selectively disable parts of the MP/M II file security mechanism.

When a requester's CCP opens a COM file for loading and subsequent execution, it examines the high-order bits of the first, second, third, and fourth bytes of the filename. These bits are referred to as interface attributes Fl', F2', F3', and F4'. The CCP constructs a byte based on the interface attribute set. It then uses this byte as a parameter for Function 70 (Set Compatibility Attributes). Function 70 causes the NDOS to send a logical compatibility attribute message to every server of which it has knowledge.

Table 3-1 defines the interface attributes.

Attribute	Meaning
Fl'	causes MP/M II to behave as though all files were opened in Read-Only mode, although write accesses are still permitted. F1' is functionally equivalent to opening a file in unlocked mode, except that record locking is not possible. Using this attribute, two programs can update the same record simultaneously, leaving the file in an indeterminate state.
F2'	causes all file close operations to convert to partial close operations. A partial close uses the current FCB to update the directory but permits the application program to continue using the file without reopening it.
F3'	disables FCB checksum verification during close operations. Files are closed successfully as long as MP/M II can tell the file was initially opened and still has an item on the system lock list. If the file was not opened, an error is still returned.
F4'	disables all FCB checksum verification. F4' implicitly sets attributes F2' and F3' as well. Use this attribute with extreme caution because it is possible to perform valid file operations using corrupt FCBs. Doing this could result in serious damage to the files on the disk drive being referenced.

Table 3-1. Interface Attributes

The CCP uses the interface attributes to construct a one-byte parameter for the set compatibility attributes call by setting the following bits:

- F1' bit 7
- F2' bit 6
- F3' bit 5
- F4' bits 4, 5, and 6

All other bits are set to zero.

The set compatibility attributes logical message causes the server to change its process descriptor if the user has enabled compatibility attributes during the MP/M II GENSYS operation. Otherwise, the message is ignored.

When an application program terminates, the CCP resets all compatibility attributes. This prevents a subsequent program from operating in an environment with insufficient file security.

It is advisable to enable the minimum number of compatibility attributes necessary to allow a program to run properly. Use the following guidelines for setting the attributes:

- If the program aborts with NDOS Error 05, FILE OPEN BY ANOTHER PROCESS, set Fl'.
- If the program aborts with NDOS Error 06, CLOSE CHECKSUM ERROR, set F3'.
- if an error code is returned in register A on I/O operations under CP/NET, but no error is returned under CP/M, try setting F2' If the problem persists, try setting both F2' and F3'. if the problem still

persists, set user attribute F4'. Make sure there is no possibility of corrupting the file system before using attribute F4'.

You can use the SET utility under MP/M II to enter compatibility interface attributes into a .COM file's directory entry from an MP/M II console. For example,

```
SET <filespec> [Fl=ON,F3=ON]
```

If you cannot use MP/M II, you can set the interface attributes under program control using Function 30 (Set File Attributes).

3.2.6 Password Protection Under CP/NET

The MP/M II file system limits file access by unprivileged users through password protection for individual files. There are three levels of password protection for files:

- All access is denied without the password.
- The file can be read without the password, but it cannot be written to.
- The file can be read and written to without the password, but not deleted.

Use the SET utility to assign passwords under MP/M II. The procedure for assigning passwords is described in the MP/M II Operating System User' s Guide. CP/NET does not support the assignment of passwords across the network.

CP/NET does, however, allow an application program to send a Password across the network when a file is opened. This allows a user on a CP/NET requester the most basic form of password support: operation on networked files that have been previously password protected.

If a read-protected file is opened and no password is specified, an extended error is returned across the network, and the Calling application aborts. The same error is also returned when an application attempts to write to a write-protected file for which no password was provided when the file was opened. Finally, any attempt to delete, rename, or change the attributes of a delete protected file without providing a password results in an extended error.

CP/NET also supports Function 106 (Set Default Password). Function 106 provides a password against which all protected files are checked if no password is provided or if the password is incorrect. This function can relieve an application of the responsibility to parse passwords constantly into the first eight bytes of the current DMA buffer.

CCP.SPR does not support MP/M II's facility of supplying passwords when the user enters a command line. Because of this, do not password-protect COM files unless a default password utility is provided to the user.

Because CP/M 2.x does not support any kind of file protection, passwords are ignored when referencing files on drives local to a CP/NET requester.

3.2.7 Networked List and Console Devices Under CP/NET

In addition to the 16 disk devices, CP/NET allows the user to map the list and console devices across the network. A number of requesters can share a printer, or a console can be logically attached to a completely independent system running CP/NET or CP/NOS. Such a system needs only a network interface to support full CP/M capability.

Unlike most requester BDOS calls, whether a console or list device is local or networked is determined, not at the BDOS intercept level, but at the BIOS-intercept level. This feature enables application programs to make direct BIOS calls for console and printer I/O and to continue to run transparently across the network.

List device I/O is handled in the following manner: when the BIOS call is made to LISTOUT, the NDOS traps it. The NDOS examines the configuration table to determine whether the list device is local to the CP/NET system or networked. If the list device is local, the call is passed through to the BIOS unchanged.

If the list device is networked, however, the NDOS stores the character to be listed in a special buffer, located directly below the requester configuration table. When 128 characters are stored, the NDOS sends a List Output logical message to the server upon which the list device is mapped. This buffering process improves system performance because one-character messages that would congest the network communication interfaces need not be sent between each requester and server.

Under CP/M, there is no need to tell the list device when a listing is complete because only one application can list at a time, and that application has complete control of the device during that time. Under CP/NET, however, more than one requester can share a printer. So a mechanism must be included to notify the server that a listing is done and that the list device is available to other requesters.

A special provision must be included so a partially filled list buffer can be flushed to the server when a listing is finished, and so the server can release the list device. Endlist, a special character equal to FF hex, is intercepted by the NDOS as the signal to terminate a listing.

The endlist character can come from one of four sources:

1. The CCP.SPR sends an endlist character every time it is entered and detects that a list is in progress. This causes an endlist every time a program terminates.
2. An application can issue an endlist to terminate its own listing.
3. Every time a CTRL-P is toggled to off, the NDOS console input routine detects this and issues its own endlist.
4. You can use the ENDLIST utility to terminate the listing.

The server can handle listing in two different modes. If the module SPOOL.RSP is present in MP/M II, the server takes all list output messages and writes them to a dedicated spooler file. When the server detects an endlist, it inserts a CTRL-Z end-of-file character into the message, closes the spooler file, and directs the SPOOL process to begin printing the file on the appropriate list device

If a SPOOL process is not resident under MP/M II, the server, upon receiving an initial list out message, performs an explicit attach list function on the specified list device. This prevents other requesters from using the list device until the requester being serviced is finished listing. All other requesters are suspended or receive network errors if they try to use the same list device. When the server finally receives the endlist character, it issues a detach list function, freeing the list device for another process.

Both server modes have potential disadvantages. A printer that uses a CTRL-Z as an escape sequence for special printing functions cannot be used with the SPOOL.RSP. Using CTRL-Z causes the spooler to terminate a print job prematurely, assuming that an end-of-file was encountered. On the other hand, explicit attaching and detaching of list devices can cause a network error if a requester attempts to attach a list device that is already in use, has its server become suspended, and eventually times out.

Console I/O cannot be buffered and sent across the network in large blocks because it is not possible to determine when input critical to the operation of an application is needed. The NDOS must therefore send such I/O across the network one character at a time.

As with list output, the NDOS traps console-related BIOS calls. The NDOS determines whether the console is local or networked. If the console is local, no action is taken, and the local BIOS is entered. If the console is networked, a raw or unfiltered console I/O message is sent to the server. The server performs the I/O function and sends a response back to the requester.

If a networked console is used with CP/NET, the system behaves unreliably when the console is also being used as a regular MP/M II terminal because MP/M II allocates a Terminal Message Process (TMP) to each known user console. Both a server process and a TMP can be waiting for input from the same console. Because of this, typed characters can be echoed normally, doubly echoed, or not echoed at all. The actual processes might or might not receive every character.

A networked console user should also be aware that, because each character must be sent over the network, networked consoles drastically degrade the performance of the entire CP/NET system. Networked consoles are not recommended unless there is no way to support a local console, as in certain industrial process-control applications.

The CTRL-P facility of CP/M is partially handled by the NDOS. The NDOS must know when CTRL-P is active because it must send an endlist character when the facility terminates. If the CCP detects that CTRL-P is active, it will not send an endlist, even if a program terminates.

3.3 CP/NET Function Extensions to CP/M

Applications accessing networked drives use the MP/M II file system to perform file operations. Many of those operations have slightly different meanings than they do under CP/M. For example, by setting the high-order bits of an FCB filename, a file can be opened or made in locked mode, unlocked mode, or Read-Only mode. CP/NET also allows an application to place a password in the current DMA buffer for opening password-protected files. Similarly, a close operation can perform either a permanent close or a partial close.

The return codes and side-effects of MP/M II functions also differ. Error-handling differences are discussed in Section 3.2.2. The open and make functions also differ. These functions return a two-byte value, called the file ID, in the random record field of the opened FCB. The file ID is necessary for performing record locking functions.

For a complete description of how individual CP/M functions work under MP/M II, see the MP/M II Operating System Programmer's Guide.

This section describes CP/NET functions that have no counterpart under CP/M. These include MP/M II functions that do not exist under CP/M, as well as a set of dedicated CP/NET functions. All of these functions adhere to exactly the same calling conventions as the rest of CP/M and all follow the same conventions regarding return codes.

FUNCTION 38: ACCESS DRIVE		
Prevents Drives from Being Reset		
	Register	**Value**
Entry Parameters	C	26H
	DE	Drive Vector
Return Values	A	Return Code
	H	Extended Error

The Access Drive function inserts a dummy open file item in the system lock list for each drive specified in the drive vector. The drive vector is a 16-bit vector in which each possible drive is presented. Bit 0 represents drive A:, bit 1, drive B:, continuing through 15 for drive P:.

The NDOS separates the drive vector into a number of drive vectors, one per server that the NDOS can find in the requester's configuration table. The NDOS then sends a logical message to each of these servers. If any of these messages result in an extended error, the function aborts.

If a server's system lock list does not have enough room to fit all the dummy items for all the drives specified, or if the open file limit for the server process is exceeded, none of the items is inserted and Function 38 returns an extended error.

Because the NDOS sends messages to each server in sequence, an extended error on one server does not indicate that servers accessed previously failed to insert open file items. This differs from MP/M II, where only one file system controls the entire lock list. Note that drives might have to be freed after a failure resulting from an access drive call.

If the NDOS is in return error mode, an error condition on function 38 causes register A to be set to 0FFH, and register H contains one of the following codes:

0A Open File Limit Exceeded
0B No Room in the System Lock List
0C Server Not Logged In

Because Function 38 is meaningless to local drives under CP/NET, no call to the local BDOS is made.

FUNCTION 39: FREE DRIVE		
Free Specified Disk Drives		
	Register	**Value**
Entry Parameters	C	27H
	DE	Drive Vector

The Free Drive function purges servers' lock lists of all items pertaining to the drives specified. The drive vector is a 16-bit vector in which each possible drive is represented. Bit 0 represents drive A:, bit 1, drive B:, continuing through 15 for drive P:.

Because dummy drive accesses, locked records, and open files are all purged, close all important files before issuing the free drive call. Otherwise, a checksum error is returned on the next file access, and data might be lost.

The CP/NET CCP issues a free drive every time a program terminates. This prevents the server process associated with the requester from becoming clogged with useless files.

Because Free Drive is meaningless under CP/M, the operating system ignores entries in the drive vector that specify drives local to the requester.

Free Drive has no error return.

FUNCTION 42: LOCK RECORD		
Lock Records in a File		
	Register	**Value**
Entry Parameters	C	2AH
	DE	FCB Address
Return Values	A	Return Code
	H	Extended Error

The Lock Record function grants a requester exclusive write access to a specific record of a file opened in unlocked mode. Using this function, any number of requester processes can simultaneously update a common file.

To lock a record, a requester application must place the logical record number to be locked in the random record field of the file's FCB. The file ID number, a two-byte value that is returned in the random record field when a file is opened in unlocked mode, must be placed in the first two bytes of the current DMA buffer. When the lock function is called, a pointer to the FCB must exist in register pair DE.

The record to be locked must reside within a block currently allocated for the file. The lock fails if the record is locked by another process or requester. This prevents two processes from simultaneously updating the same record and leaving it in an indeterminate state.

If a file was opened in locked mode, the Lock Record function always returns successfully, but no explicit action is taken because the whole file is locked in the first place.

To use the Lock Record function, follow these steps:

1. Open the file in unlocked mode. Save the file ID returned in the random record field of the open FCB.
2. When the application needs to update the record, lock the record, even before attempting to read it. Reading a record that is locked by another process can result in leaving the record in an indeterminate state. If an error results because the record is locked by another process, repeat this step until the record is locked successfully. Place a timeout value on retrying the lock in case another requester has locked the record and then gone off line.
3. Read the record.
4. Update the record.
5. Write the record back.
6. Unlock the record.

The Lock Record function returns a 0 in register A if successful. Otherwise, the Lock Record function returns one of the following error codes in register A:

01	Reading Unwritten Data
03	Cannot Close Current Extent to Access Extent Specified
04	Seek to an Unwritten Extent
06	Random Record Number Greater than 3FFFF
08	Record Locked by Another Process
0A	FCB Checksum Error
0B	Unlock File Verification Error

0C	Process Record Lock Limit Exceeded
0D	Invalid File ID in the DMA Buffer
0E	No Room on the System Lock List
FF	Extended Error

These extended errors can occur:

01	Permanent Error
04	Select Error
0C	Requester Not Logged In to Server

The Lock Record function has no meaning when a drive local to the requester is referenced. The function returns with register A set to 0.

FUNCTION 43: UNLOCK RECORD		
Unlock Records in a File		
	Register	**Value**
Entry Parameters	C	2BH
	DE	FCB Address
Return Values	A	Return Code
	H	Extended Error

The Unlock Record function releases a previously locked record, allowing it to be locked and written to by another requester. The record to be unlocked must be placed in the random record field of the file's FCB. The file ID is a two-byte value that is returned in the random field when a file is opened in unlocked mode. The file ID must be placed in the first two bytes of the current DMA buffer. Register pair DE must contain a pointer to the FCB.

The Unlock Record function returns successfully if

- the file was opened in locked mode.
- the record specified is already unlocked.
- the record is locked by another process.

In all these cases, no action is performed.

Do not unlock a record until the requester's application program has finished updating the locked record and has written it back out to the file. Otherwise, another process might inadvertently destroy the updated information.

The Unlock Record function returns a 0 in register A if Successful. Otherwise, the function returns one of the following error codes in register A:

01	Reading Unwritten Data
03	Cannot Close Current Extent to Access Extent Specified

04	Seek to an Unwritten Extent
06	Random Record Number Greater than 3FFFF
0A	FCB Checksum Error
0B	Unlock File Verification Error
0D	Invalid File ID in the DMA Buffer
FF	Extended Error

These extended errors can occur:

01	Permanent Error
04	Select Error
0C	Server Not Logged In

The Unlock Record function is meaningless when it references a requester's local drive; it returns a 0 in register A.

FUNCTION 45: SET BDOS ERROR MODE		
Defines CP/NET Error Handling		
	Register	**Value**
Entry Parameters	C	2DH
	E	Error Mode

The Set BDOS Error Mode function provides the NDOS with these options:

- aborting on extended errors
- returning the extended error to the calling application for handling
- returning the error to the application and displaying it on the console

All requester application programs are initially loaded in a default environment that causes the NDOS to abort on extended errors and to display the extended error code. Use Function 45 to change this default mode, according to the contents of register E.

Register	**Explanation**
0FFH	Return Error Mode. BDOS returns extended errors coming from the network to the application program. Register A is set to 0FFH, and register H contains the extended error code. No error message is displayed on the console.
0FEH	Return and Display Mode. BDOS returns the extended error in the same manner as in Return Error Mode, but also displays an extended error message.
Any Other Value	Default Mode.

Table 3-2. BDOS Error Modes

Function 45 is not implemented across the network. The NDOS maintains its own internal error mode flag and acts upon returning network messages according to that flag.

The Set BDOS Error Mode function has no effect on physical errors returned by the requester's local BIOS. These errors always display an error message, then they give the user the option of aborting the application program or continuing.

FUNCTION 64: LOGIN		
Initiate Session Between a Requester and a Server		
	Register	**Value**
Entry Parameters	C	40H
	DE	Ptr to Login Msg
Return Values	A	Return Code

The Login function identifies a requester to a server and initiates a session with that server. The Login function must always be successfully called before a requester can access a server's resources. Register pair DE must contain a pointer to a data structure that contains the following two fields:

00-00	Server ID byte
01-08	Password

The NDOS uses this structure to construct a logical LOGIN message to the server specified. Only the LOGIN message can be passed to the SERVER module without generating an extended error 0C, requester not logged in.

The server checks to see whether the password matches the password defined in the server configuration table. The server then scans the configuration table to find out whether logging in another requester exceeds the number of servers present in the system. If a server exists for the requester, and the password matches, the NDOS returns a 0 in register A. Otherwise, an error is flagged by returning an 0FFH in register A. The NDOS also returns a 0 in register A if the requester is already logged in.

FUNCTION 65: LOGOFF		
Terminate a Session Between a Requester and a Server		
	Register	**Value**
Entry Parameters	C	41H
	E	Server ID
Return Values	A	Return Code
	H	Extended Error

The Logoff function completes a session and breaks the logical binding between the server specified in register E and the calling requester. Once a Logoff has been performed, the server process is free to begin a session with another requester, if the the server's NETWRKIF can support the dynamic binding of requester nodes to server processes.

Function 65 returns a 0 if successful. It returns an extended error 0C, requester not logged on to server, if unsuccessful.

FUNCTION 66: SEND MESSAGE ON NETWORK		
Send a Message to Another Network Node		
	Register	**Value**
Entry Parameters	C	42H
	DE	Pointer to Message
Return Values	A	Return Code

The Send Message on Network function sends messages across the network that might have no defined function on the MP/M II server. This allows applications to be written under CP/NET that use non CP/NET messages. Point-to-point communications packages, special electronic mail systems, implementation of requester synchronization functions, and special print spooling systems are examples of such applications.

To use Function 66, the address of the message to be sent must be passed in register pair DE. The message pointed to might have the standard CP/NET structure of FMT, DID, SID, FNC, SIZ, and MSG, or it might take some nonstandard format. In the latter case, the SNIOS must be able to recognize the nonstandard message and send it properly.

Unlike the usual CP/NET session protocol, the Send Message on Network function does not automatically attempt to receive a response to the message that was sent. So an application can send throw-away messages that do not require a logical acknowledgment or response. You can also define message types that can be broadcast to every node in the network.

If an application requires a logical response to a message sent using Function 66, make an explicit call to Function 67 (Receive Message on Network).

As a rule, set the FMT field of the message header of any nonstandard message sent through a CP/NET system to a value other than those reserved for use by Digital Research. Future releases can then run applications using Function 66, with minimal modification.

Function 66 returns an FF in registers A, H, and L if a network error occurred and the message was not sent.

FUNCTION 67: RECEIVE MESSAGE ON NETWORK		
Receive Message from Another Network Node		
	Register	**Value**
Entry Parameters	C	43H
	DE	Receive Buffer Address
Return Values	A	Return Code

The Receive Message on Network function is the counterpart of Function 66, Send Message on Network. Invoke it immediately after performing a send message if a logical response is expected. Function 67 can also be used to wait for an unsolicited message from another node.

To use Function 67, an application must pass a pointer to a buffer area into which the message can be received in register DE. Upon return, registers A, H, and L are set to 0FFH if the function failed to receive the message properly.

Like Function 66, Function 67 can handle nonstandard messages across a CP/NET network, provided that the requester's SNIOS is equipped to handle them. For a more detailed discussion on how to use Functions 66 and 67, see section 3.4.

FUNCTION 68: GET NETWORK STATUS		
Get Network Status Byte from the Configuration Table		
	Register	**Value**
Entry Parameters	C	44H
Return Values	A	Network Status Byte

The Get Network Status function returns the configuration table's network status byte in register A. It also resets any error conditions in the status byte.

For a description of the fields contained in the network status byte, see Section 4.2.1.

FUNCTION 69: GET CONFIGURATION TABLE ADDRESS		
Get Configuration Table Address		
	Register	**Value**
Entry Parameters	C	45H
Return Values	HL	Table Address

The Get Configuration Table Address function returns the address of the requester configuration table maintained in the SNIOS. Using this function, an application can dynamically modify the mappings of devices across the network. The utilities NETWORK and LOCAL use Function 69 to accomplish this kind of modification

For a description of the fields in the configuration table, see Section 4.2.2.

FUNCTION 70: SET COMPATIBILITY ATTRIBUTES		
Configure Server File Systems for an Application		
	Register	**Value**
Entry Parameters	C	46H
	E	Compatibility Attribute Byte

The Set Compatibility Attributes function selectively disables the file security mechanism on all MP/M II servers to which the calling requester has networked drives. This allows certain applications that run under CP/M but not under the MP/M II file system to run under CP/NET and access networked devices.

The CCP.SPR checks the compatibility interface attributes of all COM files that it loads for execution and performs a Set Compatibility Attributes function based on the pattern it finds. This is the only time to use this function. Applications should not modify their compatibility mode in mid-execution. Doing so might produce unpredictable results.

The compatibility attribute byte is set according to the interface attributes found in the COM file's name. The following attributes cause the corresponding bits to be set in register E prior to the call to Function 70:

F1'	bit 7
F2'	bit 6
F3'	bit 5
F4'	bits 4, 5, and 6

For a complete description of how to use compatibility attributes, see Section 3.2.5.

Function 70 has no error return. Extended error messages from servers to which the requester is not logged in are ignored.

FUNCTION 71: GET SERVER CONFIGURATION TABLE ADDRESS		
Get Information About a Server		
	Register	**Value**
Entry Parameters	C	47H
	E	Server ID
Return Value	HL	Server Configuration Table Address

The Get Server Configuration Table Address function returns a pointer to parts of the specified server's configuration table. The ID of the server to be examined is passed in register E prior to calling Function 71, and a pointer to the received information is returned in register pair HL.

The data structure addressed by HL has the following format:

00-00	Server Temporary File Drive
01-01	Server Network Status Byte
02-02	Server ID
03-03	Maximum Number of Requesters Permitted on the Server
04-04	Number of Requesters Currently Logged In Bit Vector of Requesters Logged In in the Requester
05-06	ID Table
07-16	Requester ID Table

The information is identical with that contained in the server configuration table, except that the login password has been removed, and a byte containing the server's temporary file drive has added to the front of the table.

Function 71 can determine whether other requesters are logged into a server. The temporary file drive can be used when an application wants to leave a file on a server but does not know the capacity or type of the server's disk drives. The MAIL utility makes frequent use of Function 71.

The server configuration table is returned across the network in a Special buffer in the NDOS. If more than one call is to be made to Function 71, and the calls reference a different server each time, the buffer is overwritten by each successive call. If an application must examine more than one server configuration table at once the table must be copied down into a buffer defined by the application.

If Function 71 passes a server ID to which the calling user is not logged on, an extended error 0C, requester not logged in, is returned.

FUNCTION 106: SET DEFAULT PASSWORD		
Establish a Default Password for File Access		
	Register	**Value**
Entry Parameters	C	46H
	DE	Password Address

The Set Default Password function allows an application to specify a password that is checked if an incorrect password is presented during an Open File function. If a file is password protected, MP/M II first checks for a password in the current DMA buffer. If no match is found, MP/M II then checks the default password set by Function 106. If MP/M II finds a match, it allows the requested operation to succeed. Otherwise, MP/M II returns an error.

When Function 106 is performed on a requester, the requester's NDOS attempts to set the default password on every server to which a drive is networked by that requester. Since Function 106 has no error return, extended requester not logged in errors are ignored

Each server process uses an MP/M II default password slot, starting with console 0 and using as many slots as there are requesters supported.

The default password set by Function 106 persists until another default password is set.

3.4 CP/NET Applications

In addition to running standard CP/M applications packages on a CP/NET requester, you can implement special applications using the network functions available in CP/NET. The applications can handle message processing in a distributed environment. Examples include high-performance print spoolers, node-to-node transfer utilities, and network management tools.

Using Functions 66 (Send Message on Network) and 67 (Receive Message on Network) , you can define an entire set of specialized messages to provide network functions. These messages must be recognized and processed by the SNIOS and NETWRKIF, but once implemented, they can be used by application programs as though they were functions themselves.

Suppose a specific network application requires a print spooler that provides special formatting features. You can write an application program that creates messages with a special code in the format byte of the CP/NET message header. When the application wants to spool data to the special spooler on the server, it uses Function 66 to send the data.

On the server side, the NETWRKIF must be capable of recognizing the specially defined format code. When the NETWRKIF sees this format, instead of routing the message to a server process, it writes the message to a special queue. The actual spooler can reside as a process under MP/M II. The spooler reads the queue and spools the data.

Notice that Functions 66 and 67 are independent of the logical protocol of CP/NET, where every message sent by a requester implies that the requester waits to receive the message. This independence permits an application using a feature like a special spooler to return immediately after sending its message. The application need not wait for a logical acknowledgment.

Another convenient application is a file copy program that works without server intervention. Under the regular CP/NET protocol, the only way to copy a file on a local requester drive to the local drive of another requester is first to copy the file to a common networked drive, then copy it back to the other requester's drive. This is inefficient.

Instead, suppose that the users of the two requesters agree to cooperate in the copying of the file. They can do this by sending each other mail. One user invokes an application program called RECEIVE, while the other brings up an application program called SEND.

The SEND program merely reads the file into memory, then sequentially sends it to the other requester, using Function 66. The SEND program might or might not request verification from the receiving requester via Function 67. In the meantime, the RECEIVE program reads the messages from the network. No server intervention is required; only the two SNIOS modules of the requester are involved in the transmission. Even though the two requesters are only capable of sequential processing, they are still able to send and receive messages synchronously. This application does not require modifications to the SNIOS and NETWRKIF; the standard CP/NET protocol is sufficient, because such applications never reference the server.

Finally, a complex network might require automatic system monitoring and maintenance utilities. Using special message formats, you can design a set of messages that check which drives are usable on various servers, compute the best path from a requester to a given server and back, and notify the system's users of servers and requesters going on or off line. These messages can be handled automatically by the SNIOS or NETWRKIF software, or they can be implemented under the control of special application programs.

The requester's NDOS and the server's SERVER module are key components in the logical structure of the CP/NET operating system. These modules, however, do not deal with the physical problems of moving a logical message from the source requester to the destination server and back again. Implementing this task varies depending on network topology, hardware, and the characteristics of the host computer systems. These modules are therefore not portable from machine to machine. You must customize them.

This section provides the network systems implementer with the information necessary to design and implement a CP/NET system efficiently. Section 4 is divided into four parts. Section 4.1 discusses general network design issues that affect CP/NET implementation. Section 4.2 details how to implement the requester network software, the SNIOS.SPR. Section 4.3 discusses the design and implementation of the server communications software, the NETWRKIF.RSP. Section 4.4 describes the design of a CP/NET server that runs under an operating system other than MP/M II. Appendixes to this manual contain several example network communications packages.

4.1 General Network Considerations

This section explains some of the basic functions of network communications software and describes, in the most general way, how communications software fits into the overall architecture. If any of the material in this section is unfamiliar to you, consult one of the many excellent textbooks available on modern networking technology. Theoretical knowledge can help you enormously in the design and implementation of your network system.

4.1.1 Functions of the CP/NET Physical Modules

The SNIOS and NETWRKIF modules function on four levels. At the lowest level, they must handle the physical transfer of a bit stream from one network node to another. This physical layer must take into account the I/O port numbers being used for communication, the physical characteristics of the network medium, network contention schemes, and other factors.

The next layer of functions must address the problem of getting complete messages from one node to another with no errors or redundant data. This data-link layer takes the bit stream from the physical layer and processes it according to its own protocol.

If any routing from node to node is required, you must include, a network-level protocol. The network layer can be as simple as identifying when a message is destined for a particular node, or it can perform complex store-and-forward operations, compute the best route from node to node, and maintain open circuits for nodes that want to communicate.

The last layer the SNIOS and NETWRKIF must address provides an interface between the low-level communications software and the logical level operating system software. In the SNIOS, this layer must transport messages to and from the NDOS. In the NETWRKIF, the transport layer reads and writes message from and to the appropriate server queues.

The layered architecture presented here can be indistinct in implementations, with single subroutines sometimes handling all four layers at once. Figure 4-1 shows the relationship of the various layers to the network interface. Notice that the physical, data link, and network layers might have to participate in the interface to recover information to perform their functions.

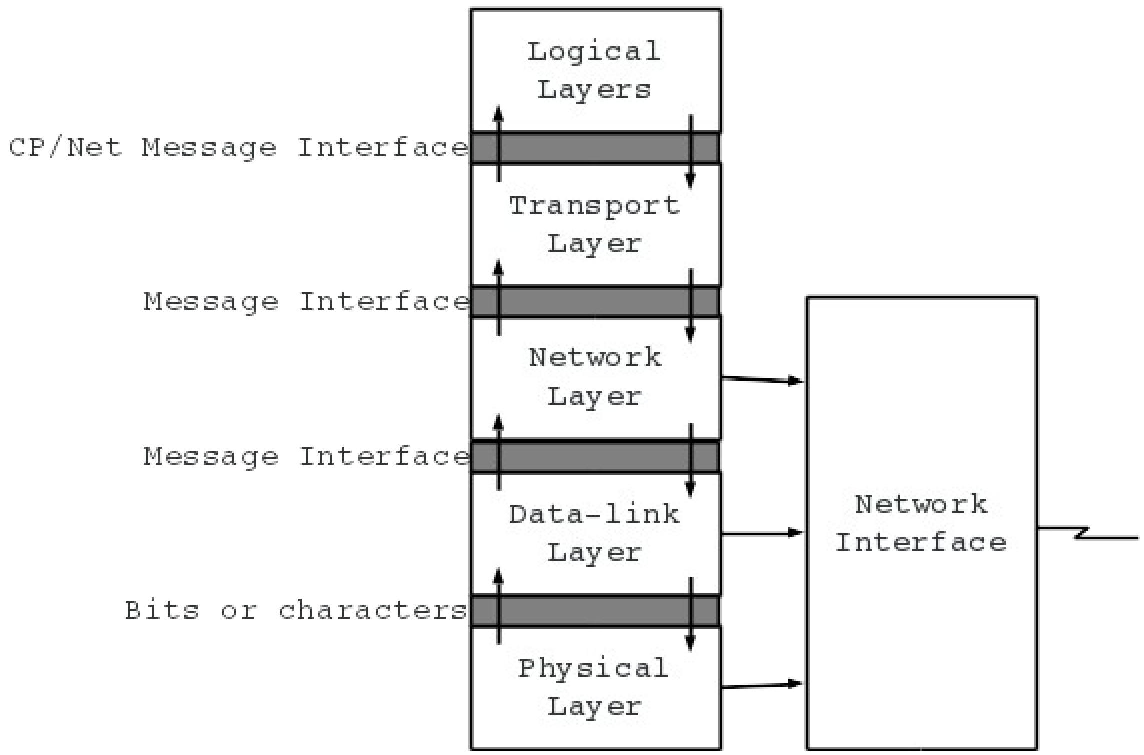


Figure 4-1. Layered Model of a CP/NET Network Node

Notice also the interfaces between the various levels. As a message migrates through the layers, the data in the message can change. The interface between the physical layer and the data-link layer yields bit or character data; the message itself is incomplete. The interface between the data-link and network layers produces messages, but the messages might contain routing information irrelevant to the transport layer. When a message reaches the transport layer, it might be in a format unusable by the higher logical layers of the operating system. only when the message is passed to those logical layers must it be complete and in the standard format of a CP/NET message.

The architecture described above corresponds to the four lowest layers of the network model described by the International Standards Organization (ISO). However, there are some slight differences. For example, the ISO definition of the transport layer concerns itself mostly with migrating messages from a centralized network controller to one of many possible hosts. In the model described above, the transport layer deals with moving messages that have already reached a host into the correct portion of the operating system. The model in Figure 4-1 is the basis for the following, more detailed discussion.

4.1.2 Interfacing a Computer to a Network

All network nodes need some method of controlling the communication functions that take place on the communications medium of the network. The simplest method is to have the node's CPU directly control all network communications protocols.

In this case, the network interface is a direct line into the host computer. When the communications software is called upon to send a message, the CPU must initiate the message, possibly waiting for an appropriate handshake response from the destination node. The CPU must then transmit the message, receive and process any acknowledgments, and determine whether the message should be retransmitted. If the node is receiving a message, it must, under program control, detect when the sender is trying to initiate a message, perform any handshake with the sender, receive the message, verify its correctness, and provide acknowledgment. All these tasks must be performed using programmed I/O operations or possibly some form of DMA for parts of the transmission or reception.

These tasks can take up a significant amount of the CPU's processing power. For an SNIOS, this is not a problem, because the NDOS is idle in the time interval after a message is sent and before the response is received. For a NETWRKIF, however, the multitasking nature of the server can result in serious performance degradation.

Another drawback to this method is that it places the burden of engineering communications software on the host systems implementer. This software can be extremely costly to develop for a high performance network.

The principal advantage of this method is its simplicity. If two computers have spare RS-232 ports, you can network them together with no special hardware. Many simple protocols can be readily modified to provide low-performance networks at low cost. Such a protocol is provided in Appendix E.

For higher-performance networks, it might be necessary to relieve the host CPU of the burden of physical, data-link, and network processing. In this case, an intelligent network communications controller can be useful. Many such controllers are available, and there is a variety of methods of interfacing them to a host computer.

An intelligent communications controller can perform all physical and data-link processing, as well as many network layer functions, with no host CPU intervention. The SNIOS and NETWRKIF modules must be concerned only with a nominal amount of network routing, if necessary, and with the problem of transporting the message from the controller. Because the communications controller can transfer data to the host at high speed with high reliability, the host's transport layer can be very simple and requires little CPU time. Appendix G provides a CP/NET implementation utilizing an intelligent network controller.

Intelligent controllers require special hardware that must be added to the host computer. Interfacing this hardware is not always possible. In addition, each network node needs a controller. This can be expensive.

CP/NET also works in multiprocessor environments, both loosely coupled and tightly coupled. A loosely coupled system can send messages via a high-speed, reliable bus. This reduces the data-link problem, so simply transferring data is often sufficient to ensure the message's integrity. Tightly coupled processors can share memory, so messages can be sent between nodes by mapping memory from one processor to another.

4.1.3 Developing a Network Layer

Because CP/NET is independent of the network used, the communication modules must be modified to support various network topologies. The NETWRKIF that supports a multidrop, contention network is different from the one that supports an active hub-star configuration.

Some CP/NET configurations require extremely complex interconnections. Messages destined for one server might have to pass unmodified through several servers or requesters before they reach their final destination. The network implementer must define the software necessary to accomplish this routing. For simple networks, a network layer is barely necessary. For example, a simple work station cluster, where several requesters share a single server, requires only that the destination ID field of the message match the server's ID on a request, and that the destination match the requester's ID when the server's response is sent back to the requester.

In complex networks, each node might need to keep track of other nodes on-line in the network. Some algorithms require the exchange of routing messages to maintain an accurate picture of the topology of the overall network. To do this, the communications software must recognize these routing messages as nonstandard CP/NET messages and not pass them to a server process or to the NDOS for processing.

Even requesters might need a network layer. For example, consider a daisy-chain network of several requesters with a server at one end. All the traffic for requesters farther down the chain passes through the requester adjacent to the server.

Because a CP/M requester can only operate a single task, the communications software for receiving and forwarding a message must be written as a series of interrupt routines. Because the NDOS might call on the SNIOS to transmit or receive a message of its own, these routines must be reentrant to the extent that NDOS requests can be held up until an intermediate message has been processed.

4.1.4 Error Recovery

Network transmission media are often unreliable. Messages are occasionally garbled or lost. In addition to data-link errors, networks can route messages incorrectly, or messages can be lost due to congestion in a section of the network. Because of these problems, a node must be able to recover from transmission errors

The most common form of error is garbled data. Bits that should have been zeros are received as ones, and ones are received as zeros. The easiest way to detect this type of error is to transmit a check along with the message. The check is computed by performing an arithmetic operation on the actual message before it is transmitted. If the check does not match the result of performing the same operation when the message is received, then a transmission error has probably occurred.

Most data-link protocols provide a mechanism for acknowledging that a message was received correctly. This mechanism requires a special message as an acknowledgment. The node that received the original message sends the special message back to the node that sent the original message. If an error occurs, the receiver either sends no acknowledgment or sends a negative acknowledgment, telling the sender to retransmit the message immediately.

The sender must be able to detect a transmission error and take steps to retransmit the message. This can be a problem because the sender does not know what the receiver is doing. If an error message comes back, the sender knows something has gone wrong. But if a message is lost completely, the receiver might not know it was sent and never send an error condition.

To solve this problem, the sender can send a message, then wait a predetermined interval for acknowledgment. If no acknowledgment arrives, the interval expires, and the sender times out. A timeout condition can cause the sender to retransmit the message or take other steps to recover from the error. When the message is finally sent successfully, the sender can free up the buffer that held it and continue with other processing.

For a CP/NET requester, two different levels of timeouts might be necessary. At the data-link level, a timeout can be set on the amount of time that elapses between sending a message and receiving the acknowledgment that it was received correctly. This timeout interval can be fairly short, since the transmission path is not likely to be very long.

The second timeout addresses the logical structure of CP/NET. Every message sent to the server implies a response to be sent back to the requester. A timeout can be set upon entering the requester's receive message routine. If the requester waits too long for a response, it can be assumed that the communication link or the server itself has crashed. With this kind of timeout, the error recovery involves much more than just retransmitting the initial message. A logical initialization must take place, probably including a CP/M warm boot.

A timeout scheme can successfully retransmit lost or garbled messages. Another problem arises, however, when the receiver's acknowledgment signal is lost. The sender, not receiving the acknowledgment, eventually times out and retransmits the message. In the meantime, the message has actually been successfully received. When the message arrives from the sender a second time, the receiver must have some way of knowing that the message is a duplicate. The receiver should ignore the message, but send an acknowledgment to stop the sender from sending the duplicate yet again.

The easiest way to detect duplicates is to assign a sequence number to each message. If the receiver does not receive the sequence number it was expecting, it ignores the message, even if the message was received correctly. Every time a message is received, the expected sequence number is incremented. Every time the sender receives an acknowledgment, the sequence number to be sent is incremented. If a message times out, however, the sequence number is not incremented.

All error recovery schemes should be free from deadlocks. A deadlock occurs when the sender is waiting for an action from the receiver, but the receiver is not performing that action because it is waiting for the sender to perform another action. Carefully analyze networks that store and forward messages from node to node for deadlocks because two nodes can try to transmit to one another simultaneously.

The means of avoiding deadlocks varies according to the network topology. A multidrop network can use collision detection. if two nodes attempt to use the network at the same time, they immediately detect that their messages are garbled and stop transmitting. To avoid continuous collisions and a consequent deadlock condition, the two nodes attempt to transmit again based on a random time interval, so that one node can start transmitting before the other.

In a point-to-point network, a properly designed message handshake can often avoid data-link deadlocks. At a higher level, enforcing a buffer allocation protocol can often prevent deadlocks. Waiting to transmit messages until the receiver has space for them minimizes the possibility of two messages continuously timing out.

4.2 Customizing the Requester's SNIOS

The communication interface between the logical NDOS and the actual network is contained in the Slave Network I/O System module, SNIOS.SPR. Because this interface varies depending on the computer system and network hardware, you must customize the SNIOS.

For most applications, the SNIOS need only be a sequential system. The SNIOS never needs to respond asynchronously to unsolicited messages. Only the NDOS must direct the SNIOS to receive messages. However, some networks require real-time response from their SNIOS modules to pass a message between two network nodes that have no direct means of communicating with one another.

This section details the design and preparation of an SNIOS for inclusion with a CP/NET requester and describes the installation of the utilities necessary to run the requester.

4.2.1 Slave Network I/O System Entry Points

The SNIOS must begin with a jump vector containing the network I/O system entry points, as shown below:

```
SNIOS:  JMP NETWORKINIT         ; Network initialize
        JMP NETWORKSTS          ; Rtn network status
        JMP CONFIGTBLADR        ; Rtn Config. Tbl Adr
        JMP SENDMSG             ; Send msg on network
        JMP RECEIVEMSG          ; Receive msg from ntwk
        JMP NTWRKERROR          ; Network error
        JMP NTWRKWBOOT          ; Network warm boot
```

Listing 4-1. SNIOS Jump Vector

Each jump address corresponds to a subroutine that performs the specific function. The exact responsibilities of each entry point subroutine are given below.

NETWORKINIT
: This SNIOS entry point is called when control is transferred to the NDOS initialization entry point after being loaded by the CPNETLDR. This subroutine performs any required network interface initialization. Initialization includes reading back-panel switches, or some other suitable source, to obtain the requester processor ID for the configuration table. If initializing messages must be sent out over the network, send them from this routine.

NETWORKSTS
: This subroutine returns a single byte in register A and determines the status of the network interface. The error bits snderr and rcverr are reset when the call is made. The format of the network status byte is shown in Figure 4-2.

```
+---+---+---+---+---+---+---+---+
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
+---+---+---+---+---+---+---+---+
              ^       ^   ^   ^
              |       |   |   |
              |       |   |   +-- snderr
              |       |   +-- rcverr
              |       +-- ctrlps
              +-- active
```

Figure 4-2. Network Status Byte Format

- active = 1 if requester logged in
- ctrlps = 1 if control P is active
- rcverr = 1 if error in received message
- snderr = 1 if error in sending a message

CONFIGTBLADR
: This subroutine returns the requester configuration table address in the HL register pair. The requester configuration table is described in section 4.2.2.

SENDMSG
: This subroutine enables messages to be sent from one processor to another via the network. The passed parameter, in registers BC, is a pointer to the message. Control is not returned from this procedure until the message has been sent. Thus, the message pointed to by the BC register pair can be modified immediately upon return. The return code, in register A, has a value of 0 indicating success or 0FFH indicating failure to access the network.

RECEIVEMSG
: Messages are received from another processor through the network with this subroutine. The passed parameter, in registers BC, is a pointer to a message buffer. Control is not returned from this procedure until the message has been received and placed into the message buffer. Thus, the message in the buffer is valid immediately upon return. The return code, in register A, has a value of 0 indicating success or 0FFH indicating failure to access the network.

NTWRKERROR
: When network errors are encountered, this procedure is called. Any required network interface device reinitialization should be performed. In typical SNIOS implementations, executing a return from the NTWRKERROR procedure results in a retry. If a retry is not wanted, an appropriate message is displayed on the console, and a warm boot is performed.

NTWRKWBOOT
: This SNIOS procedure is called each time the NDOS reloads the CCP. The sample SNIOS in Appendix E displays a

```
<Warm Boot>
```

message on the console only as a demonstration of NTWRKWBOOT. More practical applications of this procedure include interrogating the CP/NET server for messages. In this way, each time a warm boot is performed, the user is notified of messages posted for him.

4.2.2 Requester Configuration Table

The configuration table that resides in the CP/NET requester's SNIOS allows reassignment of logical devices to networked servers. The configuration table creates a mapping of logical to physical devices that can be altered during CP/NET processing. The configuration table specifies the system I/O to be accessed through the network.

The requester configuration table is defined in Table 4-1.

Offset	Explanation
000-000	Requester status byte
001-001	CP/NET requester processor ID
002-033	Disk Devices; 16 two-byte pairs, first byte high-order bit on = drive on network, with the server drive code in the least significant 4 bits; the second byte contains the server processor ID.
034-035	Console Device; first byte high-order bit on console I/O on network, with the server console number in the least significant 4 bits; the second byte contains the server processor ID.
036-037	List Device; first byte high-order bit on = list to network, with the server list device number in the least significant 4 bits; the second byte contains the server processor ID.
038-038	List Device buffer index.
039-043	List Device logical message header: FMT, DID, SID, FNC and SIZ.
044-044	List Device server list device number.
045-172	List Device buffer.

Table 4-1. Requester Configuration Table

4.2.3 Preconfiguring the Configuration Table

In many network systems, there is never any need to modify the device mappings specified through the NETWORK utility. In such systems, you can preconfigure the device mappings in the configuration table. To do this, select the devices to be networked and set the high-order bit of the first byte in the entries corresponding to those devices. Set the remote device to which the local device is to be mapped in the low-order four bits of the same byte. Finally, set the server ID of the remote device in the second byte of the entry.

Be careful when preconfiguring devices to servers that might be off line. Some CP/NET functions send messages to all servers referenced in the configuration table. If one of these servers is not capable of receiving messages, functions that might subsequently send messages to servers on line can prematurely abort.

For example, the CCP might issue a free drive function to initialize the server environment for a subsequent application program. If the previous application had left files open on two on line servers, but a third server was off line, those files are left open if the free drive message was sent to the off-line server before the on-line servers. The next application program might damage the files that were inadvertently left open.

You can solve this problem by having the error recovery in the SNIOS remove any networked device that experiences continuous timeouts, converting it back into a local device. This prevents the NDOS from making continuous references to the off-line server. A major drawback of this scheme, however, is that an application might suddenly begin referencing a local device, possibly destroying files on a local disk drive. A more secure, but less friendly protocol for dealing with off-line servers is to force a warm boot whenever a network error is encountered.

It is wise to enforce a protocol that prohibits devices from being networked until the server to which they are assigned is on line. Special utilities can be written to accomplish this by sending a dummy message to every server to which drives are mapped.

4.2.4 Sending and Receiving Messages Asynchronously

In some networks, a requester might have to receive and retransmit asynchronously a message destined for another node. For example, consider a loop network, where every node has two network ports. The network protocol specifies that all messages are sent via port #1, and all messages are received via port #2. If there is only one server in the network, but more than one requester, all messages must pass through every other requester, either as they are sent to the server or as the response returns from the server.

If a requester must asynchronously handle a communication channel, it must do so outside of the facilities provided by the single-tasking CP/M operating system. The communication protocol must be interrupt driven. An interrupt service routine must at least detect the start of a message; after that, the rest of the message can be handled sequentially or under control of additional interrupt routines. If a requester cannot support interrupts, asynchronous handling of messages might be impossible. Neither the application program nor the NDOS can periodically check for incoming messages.

A mechanism must be provided so that the NDOS, sequentially calling the SNIOS to send a message, does not collide with the asynchronous transmission of another message. Receiving messages cannot collide because only one message can come over the network at a time. To accomplish this, consider implementing the loop network described above.

As a requester's application is running, another node suddenly starts sending a message to it. The requester must now receive the message, verify its correctness, and retransmit it to another node. All of these operations must be performed without damaging the local application program. If the data-link routines do not make CP/M system calls and do not modify the message buffers used by the NDOS, the entire message can be received and transmitted transparently. When this operation is finished, the interrupt service routine returns to the application program, and processing continues. When the NDOS needs to use the network, the same data-link routines that handled the asynchronous message can be used to handle the sequential one.

It is even possible to transmit a message from the NDOS while receiving a message from some other node. To do this, the message must be able to be received a piece at a time, giving both the send and receive routines enough processor time to avoid timing out. Such a system requires a mechanism for preventing both the NDOS and the interrupt service routine from attempting simultaneous transmission. A semaphore variable can be used to control the system.

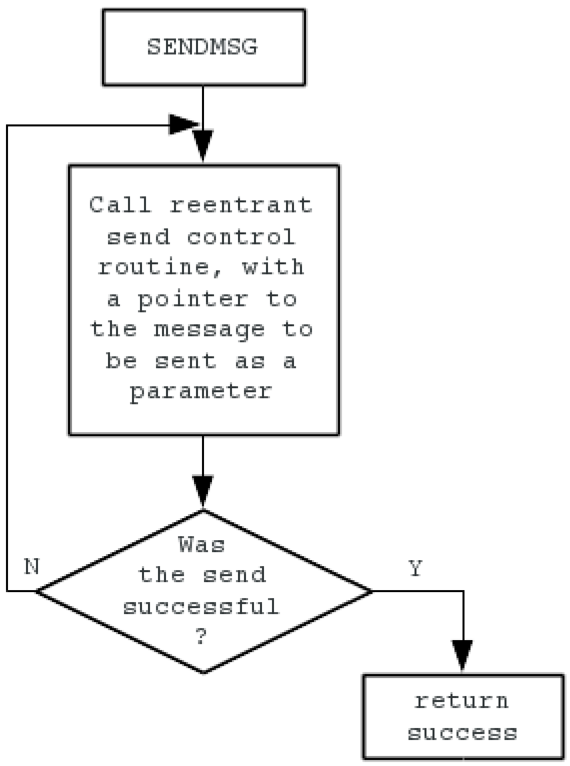
Figure 4-3 outlines a possible protocol for such a system. Both the SNIOS SENDMSG routine and the asynchronous receive interrupt service routine access a piece of reentrant code to control access to the message transmission system.

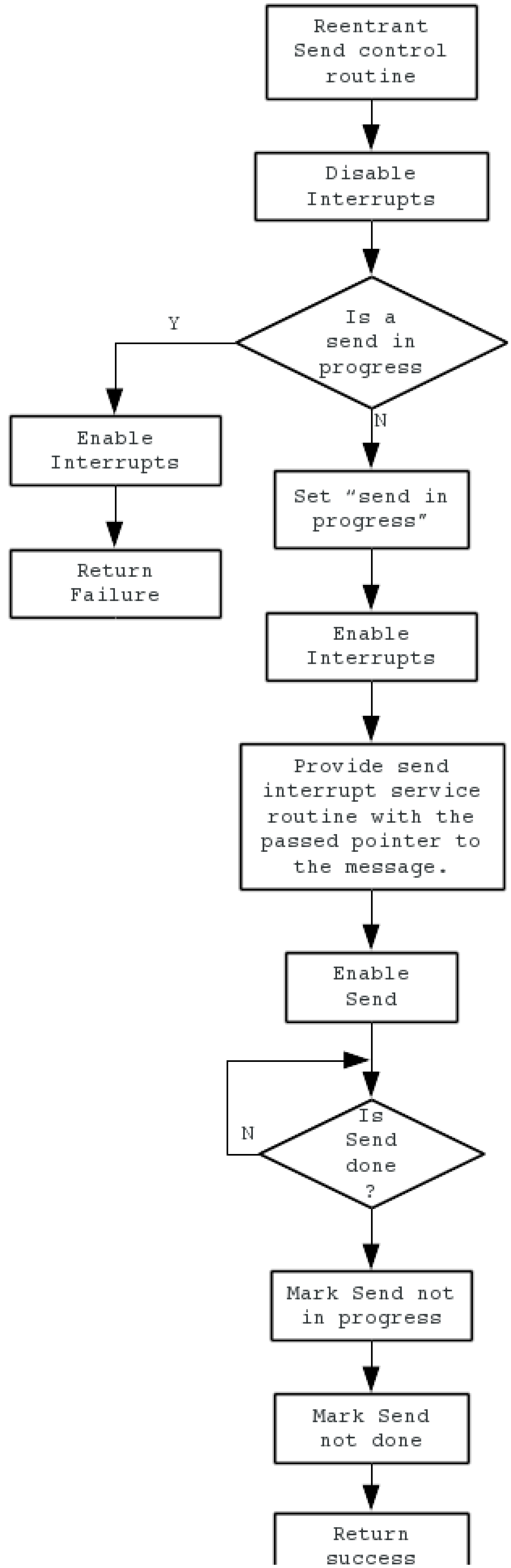
Three external events drive the system:

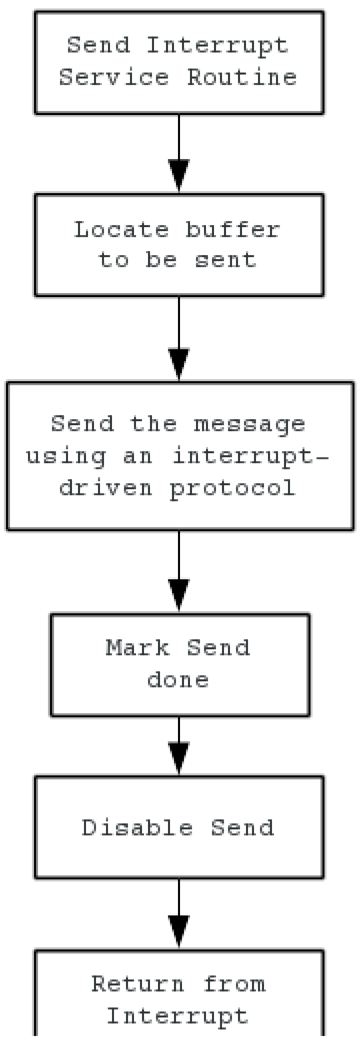
- The NDOS can request to send a message.

- The NDOS can request to receive a message.
- A message, unbidden, can cause an interrupt so that it can be received.

In this implementation, the message sending software is interrupt driven, started by enabling a transmitter interrupt. The message sending software can also operate sequentially, called by the reentrant routine that controls its use.


SENDMSG
Call reentrant send control routine, with a pointer to the message to be sent as a parameter
Was the send successful ?
N
Y
return success


Reentrant Send control routine
Disable Interrupts
Is a send in progress
Y
N
Enable Interrupts
Return Failure
Set "send in progress"
Enable Interrupts
Provide send interrupt service routine with the passed pointer to the message.
Enable Send
Is Send done ?
N
Mark Send not in progress
Mark Send not done
Return success


Send Interrupt Service Routine
Locate buffer to be sent
Send the message using an interrupt-driven protocol
Mark Send done
Disable Send
Return from Interrupt

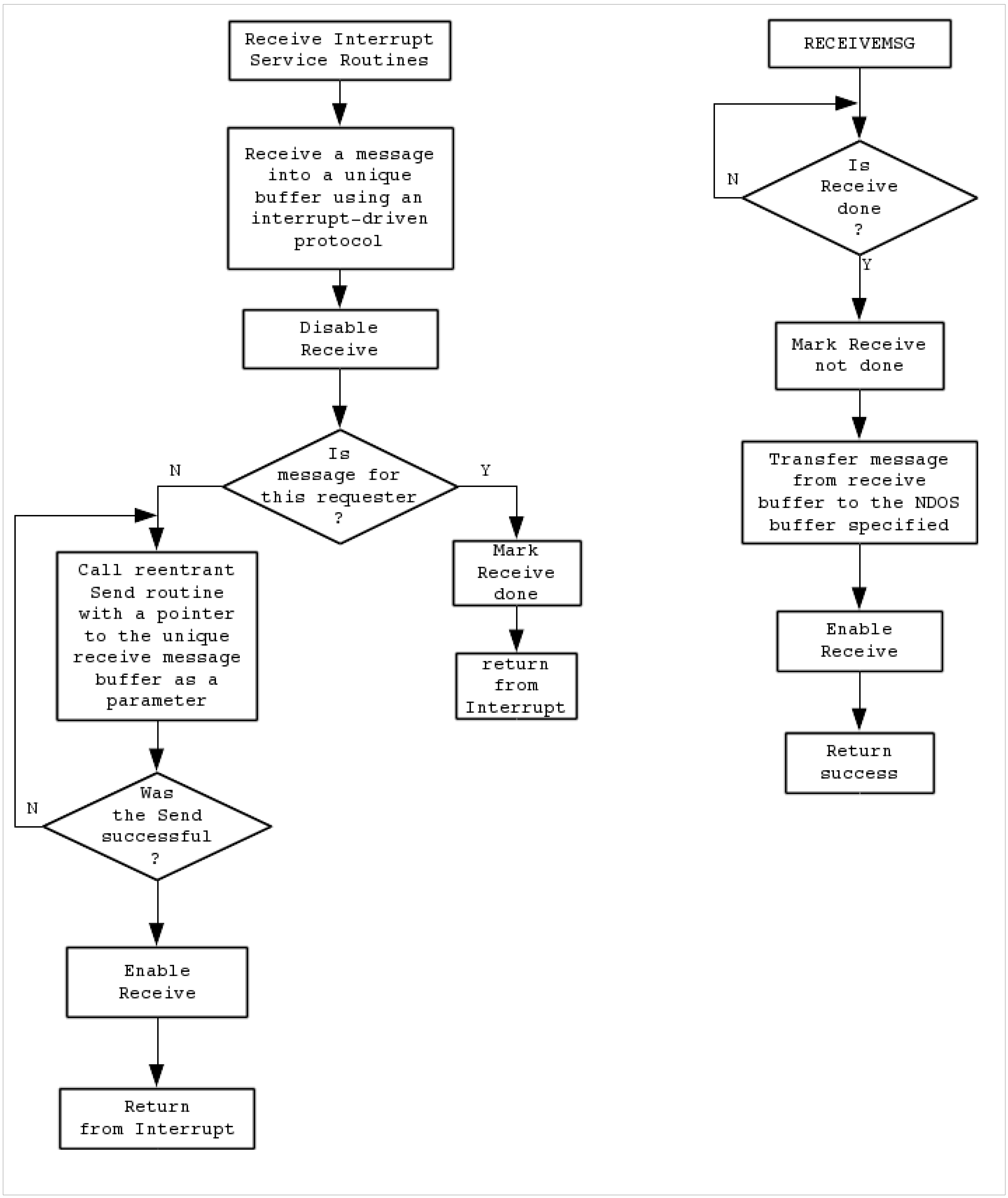


Figure 4-3. Algorithm for Interrupt-driven Requester Node that Stores and Forwards Messages

4.2.5 Generating and Debugging a Custom SNIOS

Follow these steps to generate and debug a custom SNIOS.

1. Prepare the SNIOS.SPR file, as shown below:

   ```
   A>RMAC SNIOS
   A>LINK SNIOS[OS]
   ```

   The output of the linker is the SNIOS.SPR file.

   If you do not use RMAC and LINK-80 use ASM, PIP, and GENMOD, as shown below:

   Assemble with ORG 0000H.

   ```
   A>ASM SNIOS
   A>REN SNIOS0.HEX=SNIOS.HEX
   ```

   Edit the SNIOS.ASM ORG statement. Assemble with ORG 0100H.

   ```
   A>ASM SNIOS
   A>REN SNIOS1.HEX=SNIOS.HEX
   ```

   Concatenate the HEX files.

   ```
   A>PIP SNIOS.HEX=SNIOS0.HEX,SNIOS1.HEX
   ```

   Generate the SNIOS.SPR file.

   ```
   A>GENMOD SNIOS.HEX SNIOS.SPR
   ```

   The GENMOD program uses the difference in code origins to produce a bit map of addresses to be relocated. GENMOD then places this bit map at the end of a copy of the origin 0 code and constructs a 256-byte header to create an SPR file.
2. Copy the following files to the requester:

   - CPNETLDR.COM = Loads CP/NET (NDOS.SPR and SNIOS.SPR)
   - CPNETSTS.COM = Displays status of the system I/O
   - NETWORK.COM = Redirects I/O from local to network
   - LOCAL.COM = Redirects I/O from network to local
   - DSKRESET.COM = Resets specified logical drives
   - LOGIN.COM = Logs on to server
   - LOGOFF.COM = Logs off from server
   - MAIL.COM = Electronic mail utility
   - NDOS.SPR = Network Disk Operating System
   - SNIOS.SPR = Previously Customized Slave Network I/O System
   - CCP.SPR = Console Command Processor

   you can use DDT to debug the SNIOS as follows:

   ```
   A>DDT CPNETLDR.COM
   *IB
   ```

```
*s103
0103 07 xx
*g
```

where xx is the restart the debugger uses, usually 7.

At this point, CP/NET loads, displaying the memory map, and then breaks at the specified restart. You can place breakpoints at desired locations, and then issue a G command specifying the address following the restart instruction where the CPNETLDR broke.

Communications software is difficult to debug. Because of its real-time nature, when the program is interrupted to find out what is going on, the other side of the network overruns or times out. These pointers might help you:

- Before debugging, disable any timeout logic in both the SNIOS and the NETWRKIF. This allows one node to be examined without causing errors on the other node. The SNIOS example in Appendix E accomplishes this with a conditional assembly switch called ALWAYS$RETRY.
- Never set a breakpoint in the SNIOS without setting a corresponding breakpoint in the NETWRKIF.
- Write a simulation module that mimics how you think the NETWRKIF should behave in response to the actions the SNIOS takes to send a message. Disable the actual network transmission until the SNIOS can successfully send messages to and from the simulation. Gather copious statistics because when you finally transmit over a real network link the simulation and the real NETWRKIF probably will not correspond. The statistics can help point up what was wrong with the simulation, the NETWRKIF, or both.
- Carefully verify any communications handshakes between the two nodes. You can do this by stepping through the code of both nodes simultaneously, using debuggers. Discover which data link operations can be performed while the other node is halted or disabled. Quite often, making a mistake in your debugging session points up holes in your protocol design. Once you have the protocol working with this method, have someone step one node while you step the other. Do not coordinate the actions of the two debuggers. If your protocol works without conscious synchronizing, try running it full speed.
- If possible, write one data-link module for both the SNIOS NETWRKIF, then interface them to the appropriate module. This enhances the uniformity of the protocol, making it easier to debug.

4.3 Customizing the Server

This section addresses the problems of designing and implementing an efficient CP/NET server under the MP/M II operating system. Because a CP/NET server must be capable of handling several simultaneous requests in real-time, the Network Interface module (NETWRKIF) must take full advantage of the real-time primitives of MP/M II.

The server's logical module, SERVER.RSP, consists of a set of processes, one for each requester supported. This section also discusses how the NETWRKIF sends and receives messages to and from those processes.

Finally, this section explains the system generation options available to the server implementer once the NETWRKIF has been implemented.

4.3.1 Detecting and Receiving Incoming Messages

The server is a passive, asynchronous system; it does not initiate CP/NET transactions. The server performs two distinct functions:

1. The server must detect an incoming message and initialize the communications software to receive.

2. The server must actually receive the message.

The server detects incoming messages in two ways. The first is polling, where the server periodically checks the status of the network interface. If the status changes from an idle to a ready state, the server receives a message. The second method relies on the network interface's interrupting the server. The server then transfers control to a service routine that receives the message. Either of these methods can accomplish the two functions listed above. Both methods have advantages and drawbacks.

Polling the Server

Polling is a more active method, requiring more processing overhead. If the server has a fairly heavy, continuous load of network traffic, then the status of the poll operation often indicates that a message is to be received. In this kind of system, polling has a marked advantage: the server can immediately begin receiving the message without switching contexts. But if the network traffic is subject to bursts of data mixed with periods of traffic, then the extra overhead of interrogating the network interface is inefficient.

Interrupting the Server

Interrupt driven operation is excellent for communication that occurs in bursts because no overhead is required when no communication is taking place. But very high network loads cause the server to waste a great deal of time saving the state of the process currently executing when the interrupt occurred.

Once a message has been initiated, it can be received under interrupt control, where data is processed on demand as it comes in, or under direct program control, where a process is dedicated to monitoring the incoming message. The most efficient choice depends on the type of network being used and the amount of traffic the network must handle.

In an interrupt driven communication scheme, the server responds to network events asynchronously. The network interface determines when data is processed by the host CPU. For example, when the network interface presents characters to the host, each character causes an interrupt. When the network interface performs direct memory access to transfer blocks of data, only each complete DMA transfer causes an interrupt. Depending on the protocol, each interrupt causes a specific action to be performed. The CPU is free, however, to process other tasks in between processing each piece of data. Like interrupt-driven message detection, saving the state of an interrupted process requires CPU overhead. The greater the number of interrupts required to process a message, the more system performance is degraded.

Overruns

One of the greatest problems of an interrupt-driven communications scheme develops when the interrupts occur faster than the CPU can service them. This condition is known as an overrun, and it can cause data to be lost. When an overrun occurs, the message appears to be garbled, and the sender must retransmit it. If overruns occur only when the host is extremely busy, it might be more efficient to accept the occasional garbled message in exchange for better overall response. If the number of overruns is too high, however, serious system degradation sets in. Many protocols prevent overruns by allowing the receiver to signal the sender that data is Coming in too fast.

Disabling Interrupts

The other approach to message processing uses MP/M II's facility to control processes. Unlike an interrupt service routine, which is largely transparent to MP/M II, a process is a logically complete task. Using a process-oriented protocol, you can eliminate the overrun problem by disabling interrupts while the message is being received. Disabling interrupts gives the communication program exclusive control of the CPU, so all other processing comes to a halt. If messages are fairly short, however, this method might be preferable to an interrupt-driven scheme, because no overhead is incurred by switching back and forth between a process and an interrupt service routine continually.

Selecting a Protocol

The actual data-link protocol used to process messages has not been discussed. Consider the selection of a protocol when designing how the server is going to respond to incoming messages. For example, in a CP/NET system where loosely coupled processors are communicating over a high-speed bus with little or no error checking, DMA transfer of data can be efficiently interrupt driven. But complex cyclic redundancy checks that involve extensive arithmetic operations require careful design in an interrupt-driven system, or overruns might result. Such a protocol might be better implemented using a process-oriented system.

4.3.2 NETWRKIF Module Architecture

Section 4.3.1 discusses general strategies for implementing a data-link layer protocol under MP/M II. This section deals with integrating the data-link layer into a network and transport layer. This integration allows the entire communications package to send logical requester messages to the SERVER.RSP module, and then receive the SERVER's response message for transmission back to the requester.

A dedicated server process is associated with each requester logged on to a server node. These processes are named `SERVR<x>PR` where <x> is an ASCII character between 0 and 9 or A and F. This character is a sequence number that serves as a unique identifier for the server process. Each server opens two queues that it expects the NETWRKIF module to have created. They are named `NtwrkQI<x>` and `NtwrkQO<x>` where <x> is the same character as the server's sequence number. The server process always reads the address of incoming messages from NtwrkQI<x>, and it always writes the address of the response message to NtwrkQO<x>.

This is the basic interface between the SERVER.RSP module supplied by Digital Research and the user-customized communications software. However, there are a variety of ways to implement the processes driving the interface.

Appendix E includes an example of the simplest NETWRKIF architecture. In this architecture, one network interface process is associated with each server. All processes execute the same reentrant code, but each process maintains local data that identifies the communications port it is using and the sets of queues through which it interfaces to the server process. This implementation handles its data-link software at the process level. It uses polled console I/O functions in the XIOS to detect incoming messages. This architecture is illustrated in Figure 4-4.

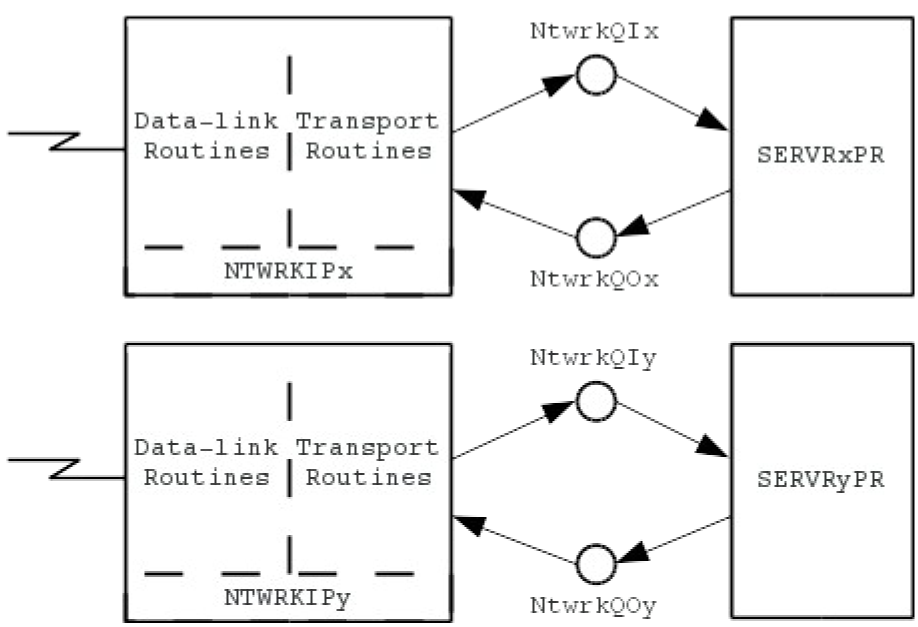


Figure 4-4. Server Architecture w/Reentrant NETWRKIF Processes

Another possible NETWRKIF architecture has only two network interface processes. An input process receives data from the network, identifies the requester that sent the message, and writes the message to the appropriate queue. An output process conditionally reads all the output queues and sends any messages it finds back out over the network.

It is also possible to force all the server processes to write their messages to a single queue by patching SERVER.RSP. In this case, the output network interface process reads the single output queue. When a message is written to it, the output process sends the message out across the network and goes back to read the queue again. An application note details how to patch SERVER.RSP. Figure 4-5 illustrates both strategies. Note that a small patch to the SERVR<x>PR processes can consolidate the output queues.

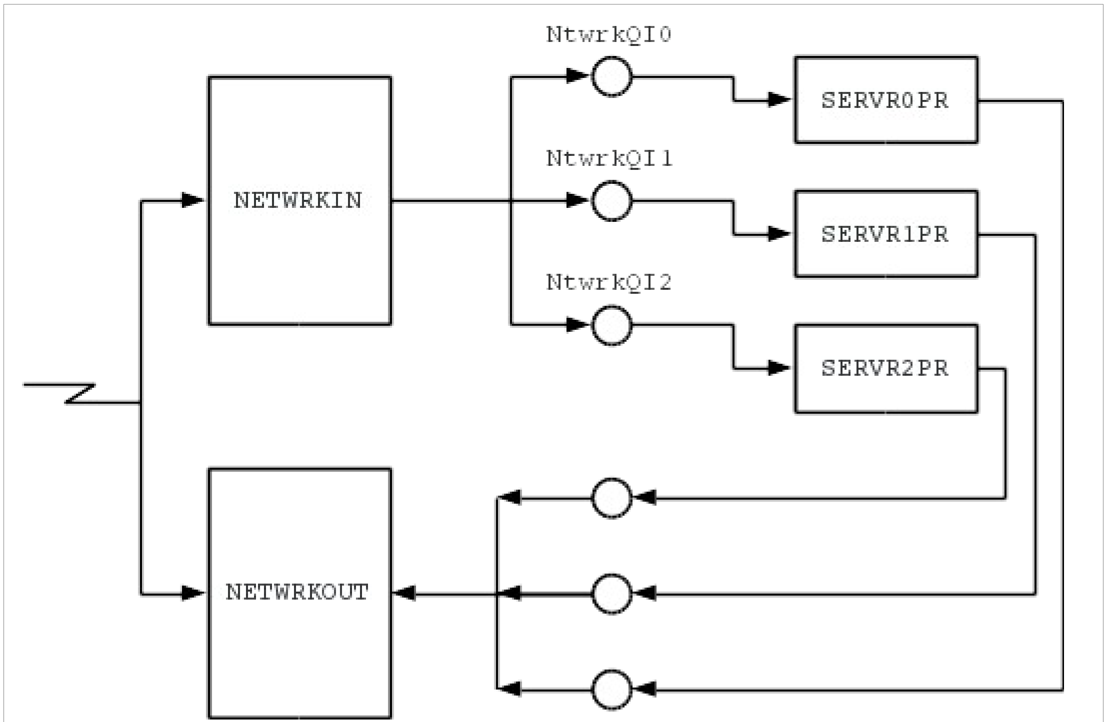


Figure 4-5. Two-process NETWRKIF

You can design a single NETWRKIF process that receives a message, writes it to the appropriate queue, then checks for any output activity. If NETWRKIF finds a message to send, it sends it, then it returns to checking for input. This kind of process has the disadvantage of being constantly busy; there is no point at which it can allow itself to become blocked. To do so might result in a deadlock or serious performance degradation.

Consider the network topology when designing the NETWRKIF architecture. For example, a NETWRKIF that uses one process per requester is suitable in an active hub-star configuration, where a unique network line is dedicated to each requester. This allows several messages to arrive at the server simultaneously.

For a multidrop topology, however, a single output and single input process NETWRKIF might be more suitable, because the network. hardware guarantees that only one message is active on the network at any one time. The same type of architecture could be applied to a loop topology.

For an active hub-star network that services several multidrop lines, it might be necessary to combine the two architectures, so that several reentrant processes are routing input to the server processes, while a set of output processes are collecting data from output queues and sending it back out of the appropriate multidrop line.

Also consider what the NETWRKIF does when it has no traffic to process. If the NETWRKIF loops madly while waiting, it will gobble up precious CPU resources, degrading the overall performance of the server system. On the other hand, the NETWRKIF must be able to respond to traffic quickly.

A number of MP/M II system calls cause a process to become blocked, so that the operating system dispatcher does not pass control back to the process until a critical condition is fulfilled. Reading an empty queue, waiting on a flag, and performing a poll call are three of the most common ways to suspend the execution of a process conditionally. Such quiescent points should be built into all NETWRKIF systems to minimize the overhead of maintaining the process when it is idle.

The processes driving the input and output queues constitute one half of a message transport layer. The NETWRKIF must also deal with how the raw message is received from the data-link and network layers that are performing the actual communication control. This interface is governed by how the data-link and network layer software is implemented.

Consider an architecture that has little or no network layer, so that the data-link software interfaces directly with the transport processes. If the data-link is included in the processes that are also performing the queuing functions, then no special interface is needed. The process can pass control from one function to another, first performing input data-link and network activities to receive a message; then computing the routing to the appropriate server input queue; then reading the response from an output queue; and finally returning to the data-link level to send the response back to the requester. The sequence can be repeated indefinitely.

Some implementations require the data-link and network layers to be under process control, with a separate set of processes controlling the transport layer. In these cases, the transport processes can use queuing for both the low-level interface to the data-link layer and the upward interface to the server processes.

This kind of architecture has the drawback of slowing down the MP/M II dispatcher with extra queuing overhead. For a small number of processes, however, the impact is slight. The architecture has the advantage of being highly modular, facilitating the future upgrade of the data-link and network layers or the transport layers. Figure 4-6 details the architecture.

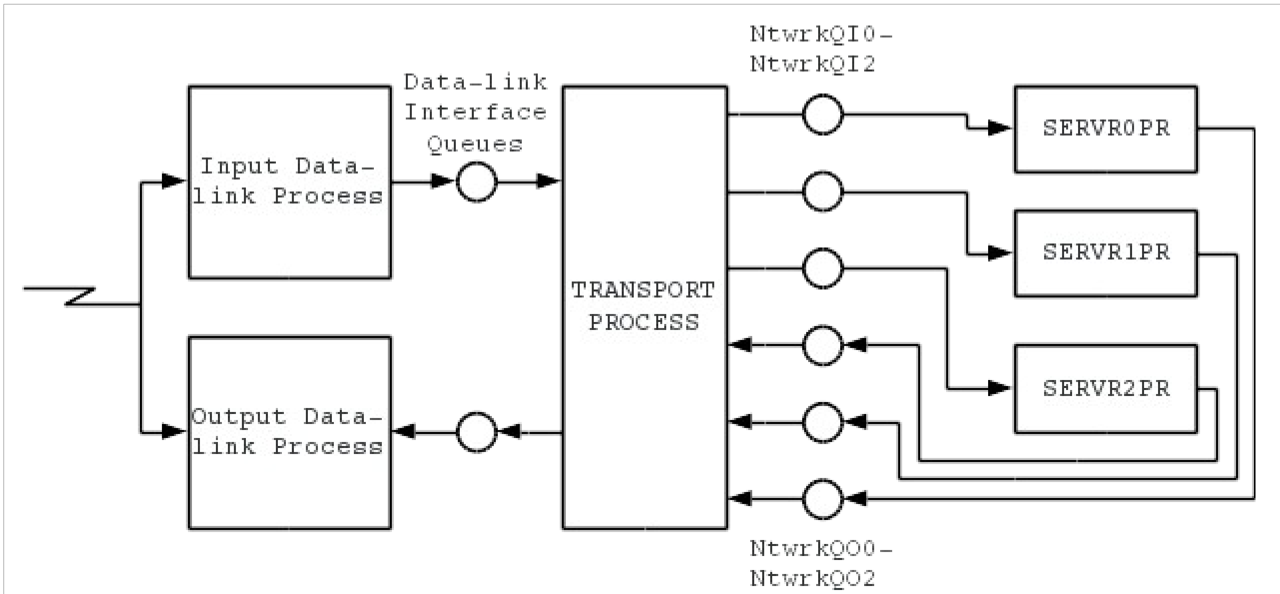


Figure 4-6. A Single Transport Process Interfacing to Low-level Data-link Processes

To implement some network interfaces, it is necessary to modify the MP/M II XIOS. Interrupt service routines must access the system interrupt vector, which is usually maintained by the XIOS. If an interface routine requires polling, the routine to accomplish the polling must be placed on the list maintained by the XIOS POLLDEVICE routine.

Interfacing to data-link and network routines that reside in the XIOS is slightly more complex than interfacing to routines contained in the NETWRKIF. These routines are often not processes, but shared code fragments or interrupt service routines. They cannot use queues as an interface mechanism. Routines that are not process-oriented must communicate through a direct function linkage, through polling, or through the Flag Set/Flag Wait functions supported by MP/M II.

Because the NETWRKIF might not be able to resolve references to such routines directly, it is often necessary to enter the XIOS through its jump vector. The XIOS jump vector table is always page aligned; a pointer to that page is located in byte 7 of the MP/M II system data page - From this point, data-link routines can be called by specifying dummy console I/O or dummy list device I/O.

If dummy console or printer I/O is used, the NETWRKIF loads a non-existent device number in register D and, if necessary, a pointer to a message buffer. The I/O routine specified checks for the non-existent device number and dispatches the call to the appropriate network routine.

Figure 4-7 illustrates how the NETWRKIF module can perform calls to subroutines resident in the XIOS.

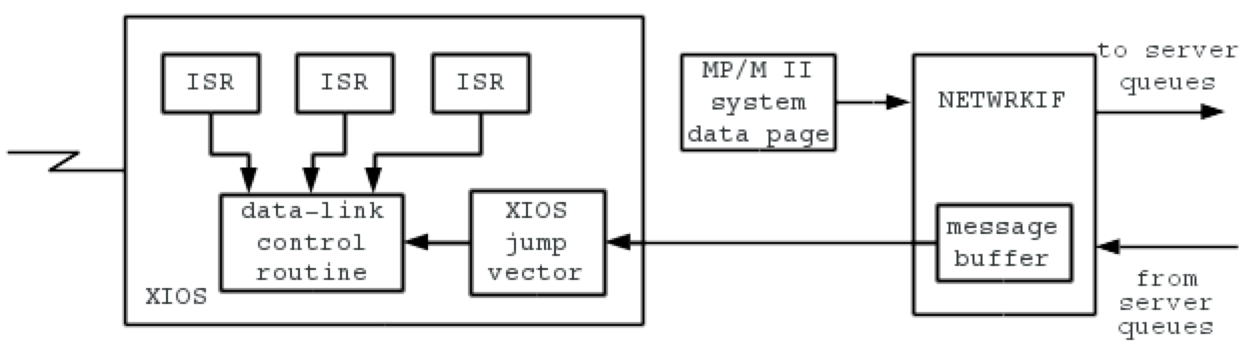


Figure 4-7. Directly Interfacing the NETVRKIF to XIOS Routines

Another method of interfacing data-link and network layer routines to a transport NETWRKIF is to have the low-level routines set a flag when a message has been processed. For example, consider a data-link routine that reads in an incoming message and checks it for validity. This routine might be a set of vectored interrupt service routines.

At this point, the NETWRKIF is not synchronized with the data link routine. When the NETWRKIF requires a message, it issues a flag-wait call to MP/M II. When the data-link routine has a complete message, it issues a flag set call. The NETWRKIF does not proceed until the flag has been set. The NETWRKIF can then transfer the message from a predefined buffer and transport it to the appropriate server process.

This type of architecture is ideal for allowing intelligent network controllers to drive the NETWRKIF transport processes. A simple interrupt service routine locates the message, builds a control block, and sets a flag to inform the NETWRKIF of the status and location of the message. Figure 4-8 shows a similar interface.

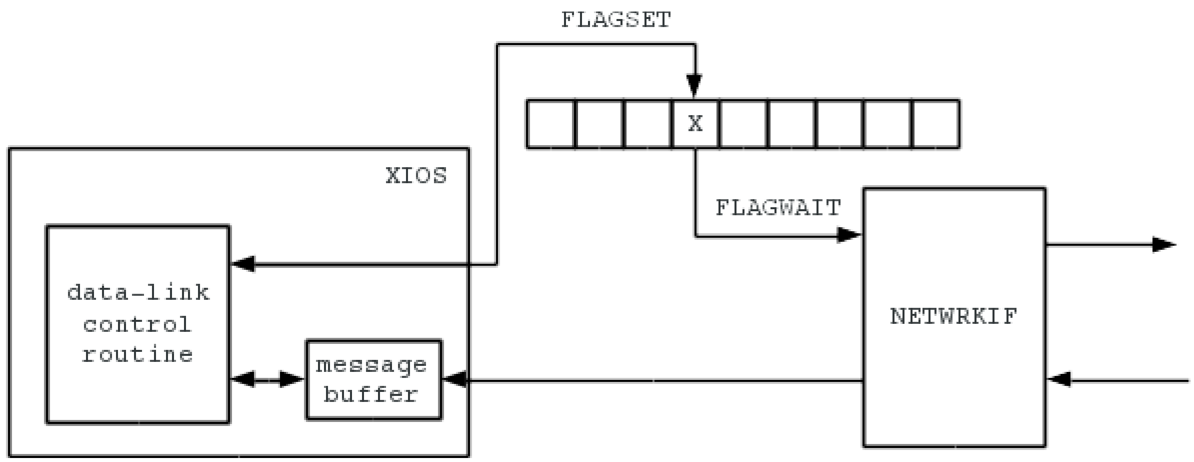


Figure 4-8. Synchronizing Data-link Activity Using Flags

To send a response message back to a requester using flags, the transport process must first identify the message to be sent and instruct the data-link layer to send it. A predefined control block can accomplish both operations. The transport process then waits on a flag until the message is sent and the flag set by the data-link.

Another possible synchronization mechanism is through the MP/M II Poll function. With this function, MP/M II suspends the calling NETWRKIF process but periodically interrogates the status of the data-link and network software through a small code fragment defined in the XIOS POLLDEVICE routine. When the status becomes true, MP/M II allows the NETWRKIF process to proceed.

If the server system supports vectored interrupts, and the location of the system's interrupt vector is known, you can write interrupt service routines that reside inside the NETWRKIF module. When the NETWRKIF performs its initialization, it simply writes the addresses of various interrupt service routines into the vector. From then on, any reference to those vector locations results in the execution of the NETWRKIF's ISRs.

This approach preserves system modularity and allows the network implementer to implement low-level routines when the XIOS itself is not available for modification. This approach still requires a synchronization mechanism between code fragments that are not part of any process and the more well-defined transport processes of the NETWRKIF.

In addition to synchronizing with low-level communications software, NETWRKIF processes might have to compete for data-link resources. For example, a transport process that wants to send a message might have to be suspended while another process is busy receiving a message. Or two reentrant processes might try to send a message out across the same network line simultaneously. These conflicts can be resolved through use of mutual exclusion (MX) queues.

An MX queue contains only one dummy message, called a token. In order to control a resource, a process must first acquire the token, leaving the MX queue empty. If another process already has the token, the first process is suspended until the second completes its resource-critical operation and replaces the token.

In this way, two low-level data-link routines--one for sending and one for receiving--can be driven without collisions by their higher-level transport processes, even if the low level routines have no explicit mechanism for sharing a network resource.

Just as the design of the network topology and error recovery schemes for CP/NET must be examined for potential deadlocks, so must the server architecture itself. A simple example of a deadlock is a process that competes for a resource using an MX queue but never restores the token to the queue when it is finished with the resource. All the other processes waiting for the resource come to a grinding halt, the network becomes congested, and eventually everything stops.

Finally, you can design an architecture that distinctly divides the data-link, network, and transport layers. The preceding synchronization strategies can be generalized to work across several layers just as easily as they can work when the server architecture divides the communications software into low-level and high-level segments. Remember that as the architecture grows more and more complex, performance of the MP/M II dispatcher and nucleus software degrade further and further. It is always wise to keep the architecture as simple as possible.

4.3.3 Elements of the NETWRKIF

This section defines the data objects and processing required to allow the server to be initialized and to operate smoothly and continuously. Through these objects, you define how many requesters a server can handle at once and how many messages can be simultaneously processed.

The following objects must be present to create the NETWRKIF.RSP module:

- XDOS entry point
- Transport Process Process Descriptors
- Transport Process Stacks
- Queue control blocks (QCBs) for the interface between the NETWRKIF and the server processes
- User queue control blocks (UQCBs) to allow the NETWRKIF to access the queues
- Message buffers
- The server configuration table
- Stack space for additional server processes, if more than one requester is to be serviced at a time
- Areas allocated to contain more server Process Descriptors, if more than one requester is to be serviced at a time
- Network initialization code
- Data-link interface code
- Message validity checking and reformatting
- Server process interface code

XDOS Entry Point

All resident system processes (RSPs) require a linkage to MP/M II's XDOS entry point because the Command Line Interpreter does not prepare an execution environment for them. This linkage is always the first two bytes of the module. When the implementer runs the MP/M II GENSYS utility to include the server modules into the operating system, GENSYS automatically fills in these two bytes with a pointer to the XDOS entry point. This allows the execution of MP/M II system functions within the body of the RSP by setting up the function parameters, loading this pointer, and dispatching.

NETWRKIF Process Descriptors

Immediately following this pointer, MP/M II expects to see a Process Descriptor. It automatically creates and executes the process to which the Descriptor refers. In the case of the NETWRKIF, this Process Descriptor controls the execution of one of the server transport processes. These processes perform the queue read and write operations to move messages into and out of the server processes. The first process must also be responsible for server and network initialization and for creating any additional transport processes.

Process Descriptors for additional transport processes must also be included, if the processes are necessary. These processes can be automatically created by linking them to the first Process Descriptor. Linking is accomplished by placing a pointer to the second Process Descriptor in the PL field of the first Process Descriptor, a pointer to the third in the PL field of the second, and so on. The chain of links terminates with a zero in the PL field of the last Process Descriptor to be created.

If you choose to have processes automatically created, remember that once processes are created, they are completely independent unless they are explicitly synchronized. The processes should not be dependent upon the first process to perform initialization for them.

Run transport processes at a very high priority, so that messages tie up the communications software for as little time as possible. The example in Appendix E runs at priority level 64, exactly the same priority as the server processes. For compute bound NETWRKIF processes, it is advisable to give the server a slightly higher priority than the NETWRKIF. The implementation in Appendix F, for example, runs at a priority of 66. This forces MP/M II always to process logical messages first if both the server and transport processes are ready at the same time.

Each transport process must have its own local stack area. Because RSPs do not have access to the extra user stack space on system calls, each stack must be capable of supporting the local storage required by the MP/M II XDOS and XIOS in addition to its own local storage.

When a process is created, its Process Descriptor's STKPTR field should point to the top of its associated stack. The top of the stack must contain the starting execution address for the process.

Queue Control Blocks

The NETWRKIF module must contain all of the queue control blocks for the entire server system. The number of QCBs varies depending on how many requesters the server system supports at one time. For each requester, there must be one input queue, named NtwrkQIO, NtwrkQI1, and so on. There must also be one output queue per requester, named NtwrkQOO, NtwrkQOI, and so on. These queues must also be created by the NETWRKIF module.

You can patch the server process code so that all processes open the same output queue, NtwrkQOO. If this patch is applied, the NETWRKIF need only include the one output QCB. The NETWRKIF examples in Appendixes F and G use this method.

The input and output queues communicate the address of the message buffer containing the message to be processed by the server or the response to be sent back to the requester. Because the message passed through the queue is only two bytes long, circular queues can be used. Both input and output queues need only buffer one message at a time because a requester must have always received a response before sending another request. Consequently, there is never more than one message from a given requester at the server at a time.

A queue capable of buffering more than one message is required only when the server processes have been patched to write all of their responses to a single queue. In this case, the queue must be capable of buffering the output from all of the servers simultaneously.

User Queue Control Blocks

Transport processes must read and write queues using user queue control blocks. These data structures contain a pointer to the appropriate QCB and a pointer to the message to be written. The queue passes only the addresses of message buffers rather than the message buffers themselves. The address of the message buffer to be accessed must be written to a location in memory, and a pointer to that location must be loaded into the appropriate UQCB.

If the UQCB can resolve the address of its associated QCB, there is no need for the NETWRKIF to open the queue using MP/M II Function 135 once the queue has been created. A pointer to the QCB can be placed in the UQCB at link time, instead. If, however, the QCB address cannot be resolved, an open queue operation must be performed. This might be the case if the system implementer breaks the NETWRKIF module into an RSP and a Banked Resident System Process (BRS).

Message Buffers

The message buffers must each be at lease 262 bytes long, 5 bytes for the CP/NET header information, and 257 bytes for the actual CP/NET message. Even though the longest CP/NET message is only 256 bytes long, the extra byte is required because the server processes use the message buffer they are passed as a temporary scratch area.

If the data-link and network layers require additional header information, the message buffers must be even longer. If the message format used by the network is different from that used by CP/NET, the message must be converted into the standard CP/NET format before it is passed to the server process. The server process expects a one-byte format code of 0, a one-byte destination code equal to the server ID, a one-byte source code, a one-byte function code, a one-byte size code, and a contiguous message in binary format. The server returns an error for any deviation from this format.

A server process always returns its response to a requester in the same message buffer that it is passed. Consequently, no transport process should modify a message in between the time that its address is written to NtwrkQI<x> and the time that its address is read back from NtwrkQO<x>. To do so can cause the server to crash.

It is not always necessary to have one buffer for every server process in the server system. Fewer buffers can be provided if the network implementer limits the number of transactions that can occur simultaneously. It is important to recognize the distinction between the number of requesters supported (the number of sessions that can be ongoing at any one time) and the number of simultaneous transactions supported (the number of messages the server can process at any one time).

Because many server processes can be idle, the number of transactions can be much lower than the number of requesters. Limiting the number of transactions can sometimes drastically improve the performance of a CP/NET server because it reduces the amount of time the operating system switches from process to process trying to service a number of file-oriented requests simultaneously.

The Server Configuration Table

The server process must interface directly with a set of objects within the NETWRKIF to perform its own initialization, maintain its own reentrant processes, and perform validity checking on its incoming messages. These three sets of objects are the server configuration table, server Process Descriptor areas, and server process stacks.

The server configuration table is defined in Table 4-2.

Offset	Explanation
00-00	Server status byte. The communications software can use this byte to signal the current state of the network. This byte has no fixed function, however.
01-01	Server processor ID. The server processes compare this field against the destination ID field of all incoming messages. An error is returned if they do not match. A server ID of FF hex is illegal. Requester utility programs use a default server ID of 0, so a CP/NET network containing only one server identifies it as node 0, for convenience.
02-02	Maximum number of requesters supported at once. Up to 16 requesters can be supported.
03-03	Number of requesters currently logged in. This field is incremented by a server process when a login takes place and decremented when a logoff takes place. Logins return an error if the maximum equals the number currently logged in.
04-05	Log-in vector. Each bit of this field indicates whether the corresponding requester ID table entry is valid and refers to a logged-in requester. When a successful login takes place, a bit is set in this vector and the corresponding table entry is updated. When a logoff occurs, the table is searched and the corresponding bit is reset.

Offset	Explanation
06-21	Requester ID table. When a requester is successfully logged in, a server process locates an empty slot by checking the log-in vector, marks the slot as used, and then writes the source ID of the log-in message into this table, using the bit vector position as an index.
22-29	Log-in password. The password sent in the log in message must match this password, or the login fails, and an error is returned.

Table 4-2. Server Configuration Table

Just as the requester configuration table can be preconfigured to map certain devices as networked, the server configuration table can be preconfigured to define certain requesters as logged in without performing a login operation.

To do this, set the current number of logged-in requesters to the number of predefined logins desired. Make sure the number is less than the maximum number of requesters permitted. Otherwise, the server's behavior becomes unpredictable.

The log-in vector should have a bit set for every requester to be prelogged in, and the requester ID table should contain the logged-in requesters. For example, for a five-requester server where requesters 1, 2, and 5 are defined as already logged in, the server configuration table might look like this:

```
configtbl:      db 0            ; server status
                db 0            ; server ID
                db 5            ; max number of requesters
                db 3            ; currently logged in
                dw 8009h        ; log-in vector
                db 1            ; requester ID table
                ds 2
                db 2
                ds 11
                db 5
                db 'WUGGA'      ; password
```

The requester ID table is position independent. When a server process checks to see if a requester is logged in, it searches the entire requester table, using the entire log-in vector to check the entries for validity. Consequently, the configuration table is not sufficient to specify the process to which an incoming message should be routed.

The transport software must maintain its own routing mechanism. For example, the NETWRKIF in Appendix E maintains its routing implicitly as local data in its reentrant processes. The example in Appendix F, on the other hand, relies on a requester control block that associates a source ID number with a UQCB.

Descriptors and Stacks

The module SERVER.RSP contains only one Process Descriptor and stack area. It is consequently initialized as only one process. SERVER.RSP must have some way of creating additional copies of itself. To do this, SERVER.RSP must know how many copies to create, and where to put the additional Process Descriptors and stacks.

By convention, the NETWRKIF process writes the address of the server configuration table into location offset 0009 in the system data page. The SERVER module uses this address to locate the maximum number of requesters from the configuration table. It then creates the maximum number, less one, of processes. To locate storage to create the additional processes, the SERVER module expects to find stack areas for the extra processes directly following the configuration table.

Server process stacks must be exactly 150 bytes long, and there Must be one stack for each additional server. For example, to support a total of five servers, 4*150 = 600 bytes of storage must he allocated after the configuration table.

The server expects the top of each additional server stack to contain a pointer to a 52-byte data area in which to create the new Process Descriptor. All of the Process Descriptor data areas must be contiguous.

Here is an example of the structure required for a four requester server:

```
server$pds:     ds (4-1)*52     ;server Process Descriptors

; (other data or code can be defined here)

configtbl:      ds 30           ;configuration table allocation
srvr$stkl:      ds 148          ;second server stack area
                dw server$pds
                ds 148          ;third server stack area
                dw server$pds+52
                ds 148          ;fourth server stack area
                dw server$pds+104
```

Listing 4-2. Stack and Process Descriptor Allocation for a Four-requester Server

NETWRKIF Execution Requirements

The initialization code must perform the following actions:

- Initialize the network hardware, or cause lower-level routines to initialize it.
- Via MP/M II Function 134, make all input and output queues required to run the server.
- Write the address of the configuration table into the system data page.

These initialization functions need not be performed by a single process; they can be distributed among a variety of processes and interrupt service routines. The address of the configuration table should be written to the system data page with interrupts disabled. This prevents the server from loading an incorrect partial address and making its process-creation decisions on invalid data.

Figure 4-9 shows a memory map, detailing how the SERVER.RSP and NETWRKIF.RSP modules fit into the rest of MP/M II, and how they communicate with one another during initialization.

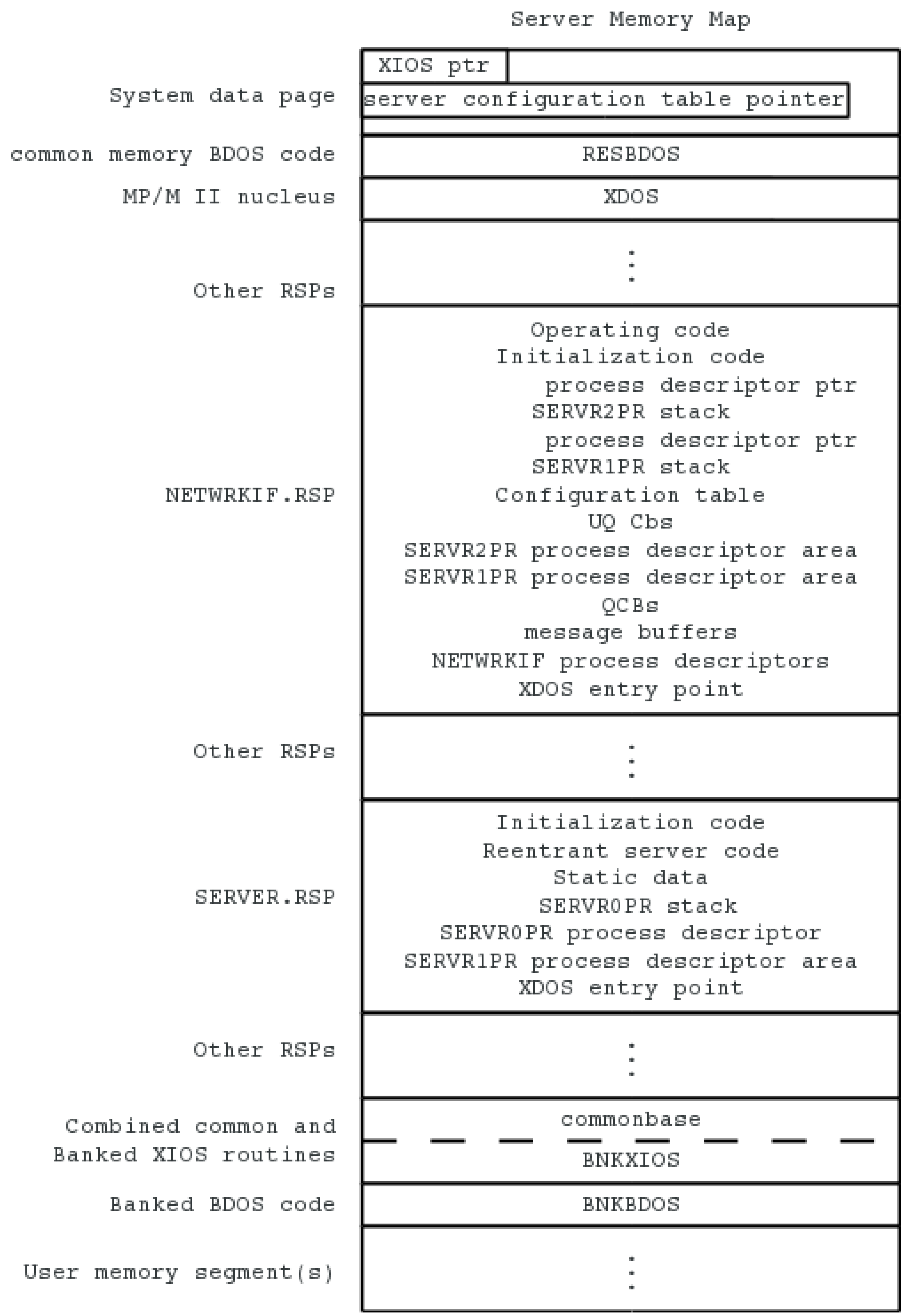


Figure 4-9. A Typical Server Memory Map

Most of the other NETWRKIF run-time functions are discussed in previous sections. The general form of the NETWRKIF is the following:

1. Allocate a message buffer and receive a message. Check the message for data-link or network errors.
2. Reformat the message, if necessary, into the standard CP/NET format.
3. Compute the server process to which the message should be routed.
4. Write the message to the server's input queue.
5. Read the response from the server's output queue.

6. Send the response back to the requester, and free the buffer.
7. Repeat this process indefinitely.

4.3.4 Enhancements and Additions to the NETWRKIF

This section deals with extensions to the basic elements required to allow a CP/NET server to run under MP/M II. These extensions can increase the capabilities and improve the performance of the basic system.

Network Initialization and Maintenance

The network interface initialization can do much more than get the server processes ready to run. In addition to passing information about the network environment to the server and physical device initialization, the NETWRKIF can interrogate the network environment to identify other nodes in the system, their status, and their resources.

For example, the NETWRKIF network layer software might send out special packets to discover on-line nodes. When other NETWRKIFs and SNIOSs detect these packets, they respond with special routing packets of their own. If these routing messages are carefully designed, each node can build a table of routes to various nodes and mark other nodes as inaccessible.

Once the network has been initialized, a special network communications process intermittently circulates the routing packets. This circulation keeps the network routing information current as nodes go on and off line.

Nodes can be interrogated to identify their system resources for networking. For example, when a process similar to the routing process just described detects the existence of a node, it logs in to the node and sends out a series of dummy select disk messages. According to the error conditions returned, the process can identify the disk drives the node has available. This can also be accomplished by having a network-layer process issue its own select disk calls in response to receiving a special message.

In implementing these schemes, make sure these special messages do not interfere with regular CP/NET traffic. Some provisions are required to ensure that requests are not made to requesters that ignore the requests or mistake them for legitimate responses to previous requests. You might have to modify the SNIOS to allow it to deal with these strange messages.

Error Handling with Timeouts

Although the transport layer software of a CP/NET system is probably extremely reliable, and the possibility of garbled messages can be ignored, network data-link errors are likely in the long run. Section 3.2.2 includes a general discussion of error handling. This section details a specific error-handling implementation, using timeouts.

Once the data-link software sends a message, it waits for an acknowledgment that the message was received. If no acknowledgment arrives, a timeout is triggered and the message is retransmitted.

You can implement a watchdog timeout mechanism as an interrupt service routine or as a process. When the transport process requests transmission from the data-link software, the process initializes a timeout variable and then waits on a flag. If the watchdog routine is implemented as an interrupt service routine (ISR), it decrements the timeout variable as a multiple of the clock interrupt frequency. If the watchdog routine is implemented as an extremely high priority process, it simply decrements the variable and then executes the MP/M II delay function for a fixed number of cycles.

With either method, a timeout status and the flag on which the transport process is waiting are set if the timeout variable is decremented to zero. At the same time, the data-link software sets the same flag and a transmission success status if it receives an acknowledgment.

When the transport process resumes processing after the flag wait operation, it checks the status variable to see which event occurred first. If the transmission timed out, the process attempts to retransmit. If the transmission succeeded, the transport process Continues.

There are many variations to this method. The preceding one assumes that the message is transmitted with no handshake or initial signal to the receiver that a message is about to follow. If a handshake is implemented, it might require a timeout of its own. Several timeout points might have to be set throughout a single message, depending on how the receiver intends to acknowledge that message.

Other error conditions can occur; they can be integrated into the error-handling structure described above. For example, the receiver can transmit a negative acknowledgment, indicating that the message was received but that it was garbled. In this case, the data-link software need only set the same event flag, but instead of setting a message received status, it sets a transmit error variable. The transport process must now differentiate between three statuses rather than two when it resumes execution, but the overall structure is the same. The architecture required to implement timeouts is shown in Figure 4-10.

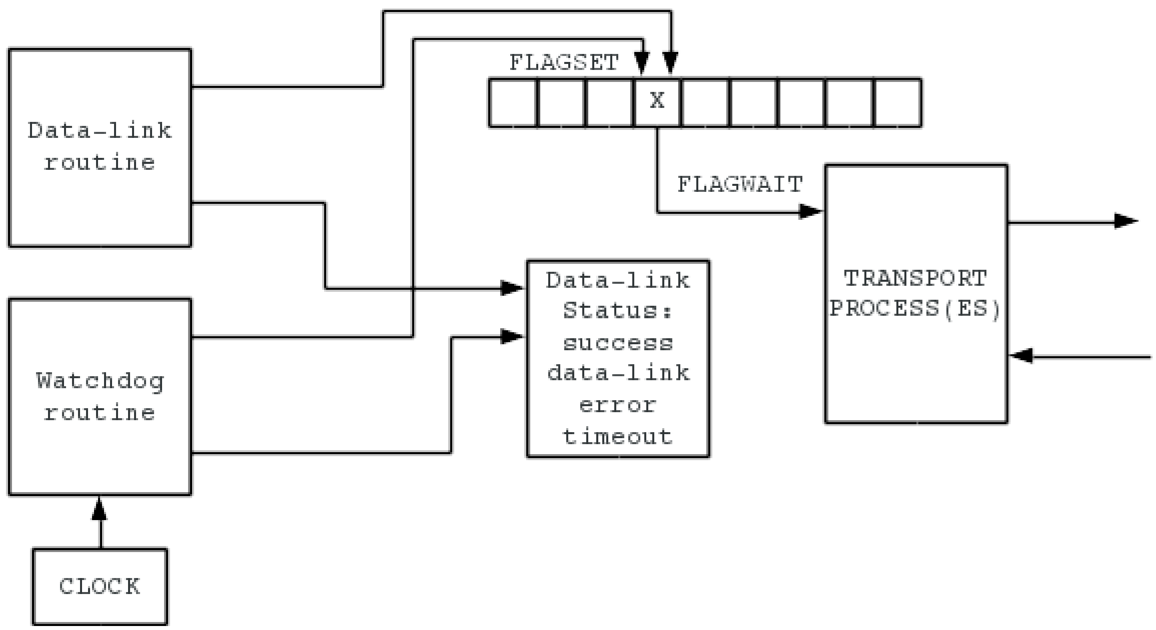


Figure 4-10. Implementing Timeouts with Flags

Store-and-Forward Networks

In some networks, the NETWRKIF can receive a message destined for another node that the sender could not reach directly. For these networks, implement network layer software to check the ultimate destination and send the message out along some other network line. These NETWRKIFs might need some of the following features.

The NETWRKIF might need more message buffers than there are supported requesters. Some messages are actually destined for the server processes resident on the current node, but a potentially high volume of the messages might be headed elsewhere.

The NETWRKIF must have a mechanism for receiving a message and then immediately sending it elsewhere without an intervening Queue Write-Queue Read operation. You can facilitate this type of operation by making the NETWRKIF software highly modular. It is advisable to have both network layer processes and transport layer processes, in addition to the data-link implementation you use. This gives the network layer process exclusive control of the data link layer, simplifying interprocess competition for the data-link resource.

Finally, the network software must have a method of knowing which nodes can be reached through which network lines. This method can be a static, predefined table or a dynamic message-passing scheme like the one described in the preceding "Network Initialization and Maintenance" section.

Dynamic Login Handling

A CP/NET server under MP/M II can handle 16 requesters at a time. Many more physical requester nodes might want to access the server. The source ID byte in the standard CP/NET message allows up to 255 nodes. Theoretically, 254 requesters can be waiting to access one server.

Obviously, it would be useful to have a method whereby a server process can be reused by another requester after its previous owner has logged off, Unfortunately, the information contained in the server configuration table is not sufficient to identify which specific server processes are free and which are in use.

To solve this problem, define one requester control block (RCB) for each requester to be simultaneously supported by the server. The RCB is defined in Table 4-3.

Offset	Explanation
00-00	Requester ID, If the control block is not in use, this field is set to FF hex.
01-03	Pointer to a particular server's input QCB.
04-05	A predefined pointer to byte 6 of this RCB.
06-07	A buffer that contains the address of the received message to be handled by this server process.

Table 4-3. Requester Control Block

Notice that this control block is a requester ID that can be matched with an incoming source ID, followed by a user queue control block. With this simple data structure, servers can be dynamically allocated to requesters with the following algorithm:

1. Receive a message.
2. Scan the RCBs for a match between the source ID of the message and the requester ID field of the RCB.
3. If a match is found, write the message buffer address into the RCB's message buffer address field in bytes 6 and 7. Then write to the queue, using the RCB's internal UQCB.
4. If a match is not found, but the scan reveals a free RCB (indicated by a requester ID field of FF) , and the incoming message is a login, then flag the RCB in use by writing the message's source ID into the RCB; update the message buffer address field; and write to the queue.
5. If a match is not found and the message is not a login, send a message back to the requester with extended error 12, requester not logged in.
6. If a match is not found, and there are no free RCBs, and the message is a login, send a message back indicating login failed.
7. When a response message is read from the queue and the message is a logoff, then free the appropriate RCB before sending the message back to the requester.

This algorithm still does not allow more than 16 requesters to be logged in at the same time. But the algorithm does permit more than 16 requester nodes to compete for access to the server node. When more than 16 requester nodes log in, they receive login failed messages. These requester nodes cannot access the server until another requester logs off. In this kind of network it is advisable to implement an automatic logoff feature for requesters that have not used the network for a fixed period of time.

Handling Special Messages

Special messages exchange network maintenance information between nodes. These messages have almost unlimited uses. For example, you can define a special message format for a special feature, high-performance print spooler. Once the format has been implemented, custom application packages can access it using Function 66 (Send Message on Network).

There are two basic steps to processing special message formats. First, the transport processes must be able to recognize special message formats and prevent them from entering the server processes. Second, the transport processes must have an interface to pass the messages off for special processing.

The first step can be accomplished by defining additional codes in the format field of the standard CP/NET message. When the transport software recognizes a strange format, it takes the appropriate action. If the message does not contain the standard CP/NET header, the data-link software can recognize this fact and notify the transport layer.

The problem of what to do with the message once it has been recognized can be solved using the same methods that are used for transporting messages throughout the more normal portions of the NETWRKIF. For example, the special print spooler and the transport layer can communicate via a predefined queue.

Some special formats require a logical response message. Functions 66 and 67 are intentionally exempt from the standard logical protocol of CP/NET. If a logical acknowledgment is required, then the transport layer must know how to accept it from the defined interface. Otherwise, the transport layer can forget that the special message occurred.

Bank-switched NETWRKIF Modules

Because of the size of the SERVER.RSP and NETWRKIF.RSP modules in a CP/NET server, MP/M II servers usually need more common memory than is available on the server system. Because of this, CP/NET users can use only one bank of their systems, completely wasting additional banks that might be used to run auxiliary processes or as additional disk buffer.

However, you can reduce the common memory requirements of an RSP by breaking it into two modules. One, still named a resident System process, contains only the code and data that must reside in Common memory to allow MP/M II to work. The rest of the module is reformatted and placed in a banked resident system process (BRS) that can be banked out when it is not executing, allowing its address space to be used by another process.

Process Descriptors and queue control blocks are the only sections of the server code that must reside in common memory. Prepare source module containing the XDOS entry point, all transport Process Descriptors, area for server Process Descriptors, all the NETWRKQIx QCBs, and all NTWRKQOx QCBs.

The first NETWRKIF Process Descriptor still must be allocated immediately after the XDOS entry point for the module, at relative address 0002H. However, this Descriptor's memory segment value should be 0FFH identifying that a BRS module is associated with it.

If any other processes exist in the NETWRKIF--for example, watchdog timeout processes--their Process Descriptors must also be included in this module. Assemble this source module and link it into RSP format. Name the object module `<netprocess>.RSP` where <netprocess> is the name of the first Process Descriptor in the module.

Then use the main body of the NETWRKIF source module to form a second source module. Remove all Process Descriptors and QCBs and place the following header at relative location 0:

```
rsp$adr:        ds 2            ;address of associated RSP
stk$adr:        dw stk$top      ;top of stack containing entry point
brs$name:       db '<netprocess>'
```

where stk$top is the address of the top of the stack for the first process, and <netprocess> matches the name of the associated RSP. This is the standard format for a BRS module; it is described in more detail in the *MP/M II Operating System System Guide*.

Because the Process Descriptors and queue control blocks are in a completely separate RSP, they cannot be resolved as simple externals. They must be defined in terms of known offsets from the beginning of <netprocess>.RSP. At run-time, the variable rsp$adr contains a pointer to the beginning of this RSP, placed there by MP/M II's GENSYS utility. Using this pointer and the predefined offsets, required references to these data objects can be resolved.

On startup, the NETWRKIF processes perform the following initialization:

1. Initialize the stack pointer fields in all NETWRKIF Process Descriptors with a pointer to the top of the stack associated with each process. This is not necessary for the first process because GENSYS provides the stack pointer linkage via the header data in the BRS.
2. The make queue operations the NETWRKIF requires can be complicated because the QCB addresses must be resolved. Once they are, however, update the UQCBs associated with them with those addresses, avoiding the necessity of performing open queue functions.

The NETWRKIF.BRS module requires a different way of referencing the operating system because it does not contain a pointer to the XDOS entry point. The RSP associated with the BRS module, however, does contain such a pointer as its first two bytes. The following subroutine performs operating system calls transparently:

```
do$os:  lhld rsp$adr
        mov a,m
        inx h
        mov h,m
        mov l,a
        pchl
```

you must also assemble this module and link it into RSP format; but name it <netprocess>.BRS.

Banking out the NETWRKIF module alone might raise the BNKXIOS COMMONBASE entry point above the hardware bank-select point, allowing banked operation of MP/M II. If banking out the module does not accommodate this, you can use a patch to convert SERVER.RSP into a banked module in a similar way. The patch is detailed in *CP/NET V1.2 Application Note #2, 11/11/82*.

Perform GENSYS with a specified banked system. You can add memory segments to occupy the new banks. The address ranges of the new memory segments are prompted for at the end of GENSYS.

If the number of requesters to be supported still requires more common memory than is available, there is no purpose in implementing a banked version of the server.

A banked-out server has a marginally slower response time because the dispatcher must select the system bank and because of the added level of indirection in calling the operating system. This degradation, however, is insignificant.

Although banking out the server provides additional user Segments under MP/M II, resist the temptation to add additional consoles to the system. Because of the extremely high priority at which the server runs, performance on additional consoles is very poor. However, these extra banks do provide the user with a means of performing occasional jobs directly from the MP/M II level. more importantly, extra segments can enhance the server itself by using special CP/NET messages.

4.3.5 MP/M II Performance Factors Affecting the NETWRKIF

The characteristics of the network for which a server is being implemented influence the architecture of the NETWRKIF and the rest of the server software. Another important factor in designing efficient servers under MP/M II is the nature of MP/M II itself. This section points out the overhead MP/M II incurs in implementing multitasking programming environment.

The heart of the MP/M II operating system is its dispatcher. This routine is entered every time a system call is made. The dispatcher protects system resources, tests for events that could influence the execution of any process in the system, and finally chooses the processes to execute and their order. The dispatcher takes roughly 900 microseconds to execute, but interrupts are disabled for no longer than 90 microseconds. This overhead is incurred on every system call.

The limitations of the dispatcher alone place some basic constraints on communications speed. If the network is using a serial I/O device capable of buffering three characters at 10 bits per character, then the NETWRKIF had better not rely on a system call like console input to receive network messages if the transmission rate is faster than 33K bits per second and the sender sends characters as fast as possible. Even below this speed, overruns are likely if there are any other processes in the system. This assumes an extremely simple protocol. If the network has extra signal lines, most serial I/O devices permit the receiver to signal a clear to send condition back to the sender. But networks often must work without these extra signals.

Because interrupts are disabled for no longer than 90 microseconds, a network that works at the character-interrupt level functions properly at transmission speeds up to 333K bits per second. Beyond that speed, overruns are likely to occur too often for adequate performance.

At speeds higher than 333K bits per second, the network interface software can use one of three approaches:

- A process can disable interrupts and perform no system calls, preventing the dispatcher from being entered, and perform its own direct network 1/0, character by character.
- The network interface can use DMA to transfer large blocks of message data and perform validity checking after the message has been transferred.
- The network interface can use an intelligent protocol controller that also does DMA or it can map completed messages from its own memory space into MP/M II's memory space.

Serial I/O is not the only possible network transmission medium. The example is provided to acquaint you with the performance of MP/M II.

The amount of time spent in the dispatcher varies depending on a number of factors. Because the dispatcher must check suspended processes against system events, keep the number of processes, queues, flags, and poll calls to a minimum. Poll calls are especially degrading. Every time the dispatcher is entered, it executes every code fragment associated with every outstanding poll call. If all 16 reentrant NETWRKIF processes polled output ports at once, the dispatcher would be very busy. In fact, enough poll calls can lengthen the dispatcher's execution time so much that it exceeds the clock interval. When this happens, the dispatcher is reentered before it has even been exited.

The design of interrupt service routines Must take the structure of the dispatcher into account. ISRs must first of all save the register image of the process they interrupted--the service routine then executes. When the ISR terminates itself, it should restore the interrupted process's registers and take one of two actions:

- If the service routine winds up setting a flag, the flag set call to MP/M II should be made, followed by a jump into the dispatcher. This allows the dispatcher to ready the process waiting on the flag as quickly as possible.
- If no flag is to be set, the ISR can simply return to the interrupted process.

ISRs should perform no MP/M II system calls except for the Flag Set function. There are two reasons for this. First, ISRs are not processes, so the dispatcher has no way of saving the status of the ISR in a Process Descriptor before allowing the function to be performed. Second, the dispatcher reenables interrupts and possibly dispatches another process, leaving the ISR and the interrupted process in an indeterminate state. The Flag Set function is specifically recognized by the dispatcher to avoid dispatching a different process.

Several factors determine how often the NETWRKIF and server processes are dispatched. The most obvious is, once again, the number of processes. If MP/M II must share the CPU with more tasks, there is less CPU available. Consider the priority of the various network server processes carefully. All processes in the SERVER module run at a high priority level of 100. processes in the NETWRKIF might require other priorities. In general, assign compute-bound processes lower priorities than I/0-bound processes, to prevent processes that perform few system calls from hogging the CPU.

The dispatcher always schedules processes according to priority. Improperly setting priorities can cause processes to be permanently suspended. For example, consider a NETWRKIF module that performs all direct I/O and busy-waits for network input. Suppose this process has a priority of 60, slightly higher than the server processes. Although the dispatcher is entered every time the system clock ticks, the NETWRKIF is ready. Because the NETWRKIF has a higher priority than the server processes, the server processes never execute.

Note that because of the extremely high priority of the server process, normal user processes running under MP/M II perform very poorly. In addition, the extra process load degrades the server performance. It is recommended that normal work station terminals not be provided on an MP/M II system that is functioning as a server, although a system console can be convenient for monitoring system performance and giving the operator a means of maintaining the server's data base.

The last factor affecting the dispatch rate is the system clock frequency. Every time a clock tick occurs, the dispatcher is entered and recomputes the process to be executed next. Processes of equal priority are dispatched on a first come, first served basis. The system clock can be tuned for optimal network performance . There are no general rules on tuning because each network and the applications run on the network determine the optimal clock period. Experiment with the clock frequency to determine the best performance for the server.

In addition to designing the NETWRKIF for the server system, you might want to reexamine the XIOS used in the system. Many CP/NET users discover that once their communications system has been optimized, server performance has improved only slightly because several requesters are forcing the disk system to thrash.

Thrashing can be minimized if the XIOS is provided with efficient blocking/deblocking algorithms like those discussed in the *MP/M II Operating System System Guide*. These algorithms buffer disk accesses, deferring physical Read-Write operations until they are absolutely necessary. As a result, many file record Read-Write operations occur at memory speed, instead of having to wait for physical I/O from a disk drive.

Extra blocking/deblocking buffers can also improve overall server performance enormously. Because a dedicated server only requires a single tiny user program segment, or, in some cases, no user segment at all, almost all additional memory remaining after the server has been implemented can be used for disk buffers. In a bank-switched or memory-managed system, potentially huge amounts of memory can be made available for disk buffers. Providing one or more disk buffers per supported requester potentially eliminates competition between two requesters for buffer resources.

Another way to improve disk performance with limited memory for disk buffers is to limit the number of transactions that can be present in the server at one time. Even if a server is supporting 16 requester sessions, it is possible, for example, to permit only four or five messages to be active at a time. This limit reduces the amount of competition between actual processes, although competition continues between individual transactions. Quite often, however, the overhead incurred by refusing network messages and forcing requesters to retransmit them is considerably less than the overhead incurred by repeatedly having to flush disk buffers for use and reuse by individual processes.

You can estimate the average number of disk accesses an application program is likely to perform in a short time. The NETWRKIF processes can then selectively transport messages from only one requester for a short amount of time, then service another requester for an equal amount of time. The scheme allows a single process to take maximum advantage of the blocking and deblocking algorithms implemented in the server's XIOS. The major disadvantage of such a scheme is that it is extremely complex and difficult to implement efficiently. Carefully tuned, however, it can greatly improve performance.

4.3.6 Generating the NETWRKIF

To create the MP/M II server, perform the following steps:

1. If the XIOS has been modified, generate a new version of RESXIOS.SPR or BNKXIOS.SPR or BNKXIOS.SPR, according to the instructions provided in the *MP/M II Operating System System Guide*.
2. Assemble and link the NETWRKIF module:

   ```
   A>RMAC NETWRKIF
   A>LINK NETWRKIF[NR,OR]
   ```

   The linker generates the NETWRKIF.RSP file.

   If RMAC and LINK are not available, you must use ASM, PIP, and GENMOD, as shown below:

   Assemble with ORG 0000H.

   ```
   A>ASM NETWORKIF
   A>REN NTWRK0.HEX=NETWRKIF.HEX
   ```

   Now edit the NETWRKIF.ASM ORG statement to locate the module at 100 hex. Assemble with ORG 0100H.

   ```
   A>ASM NETWRKIF
   A>REN NTWRK1.HEX=NETWRKIF.HEX
   ```

   Concatenate the HEX files.

   ```
   A>PIP NETWRKIF.HEX=NTWRK0.HEX,NTWRK1.HEX
   ```

Generate the NETWRKIF RSP file.

```
A>GENMOD NETWRKIF.HEX NETWRKIF.RSP
```

3. Copy the following files to the server boot disk.

    - SERVER.RSP = Server process Module
    - NETWRKIF.RSP = Custom Network Interface Process
    - MAIL.COM = Mail Utility
4. Perform a GENSYS on the MP/M II system. The GENSYS must include the SERVER.RSP file and the customized NETWRKIF.RSP; it can also include the SPOOL.RSP.

When GENSYS asks for the number of consoles, do not include the consoles (character I/O drivers) that support the requesters. Usually, the response is 1.

You must also configure the file system for the types of applications CP/NET runs, enable compatibility attributes, if necessary, and so on. These issues are discussed in the *MP/M II Operating System System Guide*.

4.3.7 Debugging the NETWRKIF

The MP/M II server is now ready to be debugged. There are three general strategies for debugging the server.

Debugging MP/M II Under CP/M

To debug MP/M II under CP/M, follow these steps:

1. GENSYS the MP/M II with the top of memory set below where a CP/M system running on the same hardware would reside when it is running DDT, SID, or ZSID.
2. Boot up CP/M on the server target computer system.
3. Run MPMLDR under the debugger. You can halt the loader just before passing control to MP/M II through the following sequence:

```
A>DDT MPMLDR.COM
*I$B
*G
```

When the loader breaks, you can insert breakpoints and restart the loader.

When using this method, remember that, because CP/M is a single-tasking operating system, the entire CP/M operating system becomes part of the process in which a breakpoint is inserted every time the system encounters a breakpoint. Furthermore, DDT and SID reenable interrupts on breakpoints. If a clock tick goes off, the MP/M II dispatcher is likely to suspend CP/M and continue with other processing. This might not inconvenience you because the process that was breakpointed is also suspended. If it does affect the operation of the system, you might have to disable the system clock.

Debugging the NETWRKIF as a COM file

The example in Appendix E is set up to debug the NETWRKIF as a COM file. Debugging instructions are also included in Appendix E.

Inserting Trace Code Into the NETWRKIF

Gather run-time statistics by inserting trace code into the NETWRKIF. Although this is not very helpful for debugging real-time problems, it is the least destructive method of gathering real-time statistics. This method can also be useful when tuning the network for increased performance.

4.4 Implementing Non-MP/M II Servers

It is possible to implement a CP/NET server on any computer system, under any operating system. There are several reasons why you might choose another operating system:

- MP/M II servers limit the number of requesters to 16. You might want more than 16 work stations to have access to a common database.
- You might require higher performance levels. The high speed of a mainframe CPU can substantially increase CP/NET performance.
- You might want your system to take advantage of the large base of CP/M applications programs, but maintain its files under another operating system. Or you might want to create a gateway to one of the other commercially available network systems. A special server could translate CP/NET messages into an appropriate format for the other network.

The module SERVER.RSP cannot be used on a different processor or under a different operating system. So you must not only create the equivalent of the NETWRKIF for the target computer system; you must also write the logical portion of the server.

The server processes under MP/M II act essentially as a proxy for the requester assigned to them. For example, the requester wants to open a file on a networked drive but it does not have access to the operating system controlling that drive. Instead, the requester sends a message to a server process that does have direct access to the controlling operating system and asks that process to open the file for the requester. The server obligingly performs the operation for the requester and tells it what happened. This is often referred to as a ghosted process model of a server because the operating system thinks it is running the entire application program as a process, while in fact the application is running somewhere else, but has a friend to help out.

Using the logical messages included in this manual, you can write a ghosted process server for CP/NET under almost any multitasking operating system. You can even write a CP/NET server under a single-tasking operating system. (CP/NET servers have actually been implemented under CP/M.)

The basic elements of such a server are

- A communications interface.
- A function interpreter. This module must interpret the logical messages sent by the CP/NET requester and take the appropriate action.
- A file system translator. This module must convert CP/M BDOS File Control Blocks passed by the requester into native operating system File Control Blocks.
- An operating system interface. This module must translate a CP/NET function that corresponds exactly to a function supported by MP/M II into a function or set of functions supported by the native operating system.

Each of these functional modules varies depending on the environment under which it is forced to execute. The communications interface is governed by the types of process architectures the target operating system can support. The remaining modules can be a set of reentrant processes, as they are under MP/M II, or they can be a single process that keeps track of the requester it is currently servicing. If the latter method is used, the server must keep track of such context sensitive information as directory search first/search next information and shared files.

It might not be possible to support all CP/M functions under a non-MP/M II server. If this is the case, choose applications that do not require the use of the unsupportable functions.

Finally, it might be necessary to have several different computer systems and operating systems acting as servers in the same network. It is best to make the server implementation as portable as possible. Implementing the server in a high-level language is a first step to portability.

Making the system highly modular can improve its portability. For example, break the communications interface into a hardware interface module, a data link module, a network module, and a transport module. All of these modules, with the exception of the hardware interface, can port to different systems with minimal modification.

The server's function interpreter should be completely portable, but you will probably have to rewrite the file system interpreter and the operating system interface modules.

Appendix A
CP/NOS Overview

A.1 overview

CP/NOS is a version of the CP/M operating system that performs all file handling across a CP/NET network system. CP/NOS supports one local console and one local printer, but it supports only remote mass storage media. Because of this, the BDOS and BIOS modules in a CP/NOS system are considerably smaller than their counterparts in a standard CP/M system. This allows CP/NOS to fit in a fairly small (usually 4K bytes) Read-Only memory, so you do not need a bootstrap loader. CP/NOS can also be downloaded from a server. Using a small loader, you can also download a CP/NOS system from a centralized server.

Programs written under any CP/M 2.x system are fully compatible with a comparable CP/NOS system, provided that mass storage devices referenced by the application are available across the network. When BDOS calls that service, these devices are automatically translated into network functions.

Unlike CP/NET, CP/NOS cannot be loaded under an existing CP/M system. The network modules and CP/M modules must be linked together and executed in a stand-alone environment. The special problems this creates in debugging CP/NOS are discussed in this appendix.

A.2 System Requirements

CP/NOS can run on an 8080, 8085, or Z80 microprocessor, with a maximum of 64K of memory. A usual CP/NOS system can be placed in a 4K ROM.

The CP/NOS requester must be networked to an MP/M II server. The server is the same as the one used by CP/NET. CP/NOS and CP/NET requesters can even be networked to the same server.

A.3 Customizing CP/NOS

Three of the modules incorporated in CP/NOS are system dependent and must be modified to work on a particular hardware configuration. They are the CPBIOS, CPNIOS, and NETWRKIF modules

The CPBIOS can be exactly the same as the BIOS used in a CP/M system that runs on the same hardware, except that only a small portion of the BIOS is required. The only routines required are:

BOOT	cold start
CONST	read console status
CONIN	read console character
CONOUT	write console character
LIST	write character to the list device
LISTST	read list device status

The CPBIOS jump vector must be the same as that of a regular BIOS, but all other entry points can be null.

The CPNIOS module takes the place of the SNIOS module in CP/NET and requires only minimal modification. The only difference is that all variables must be initialized upon cold start, including the requester configuration table. The utilities NETWORK and LOGIN are not sufficient to define the configuration table after cold start because CP/NOS has no local disk drives from which to load these utilities. The CPNIOS must also prompt the user for login information upon cold start, or a warm boot results in continuous requester not logged in extended errors as the CP/NOS requester tries to load the file CCP.SPR from a server that has no knowledge of the requester.

The SNIOS example in Appendix E contains a sample CPNIOS, conditionally assembled out. To obtain the CPNIOS version, equate the literal CPNOS to true.

Note: if the two preceding routines are to reside eventually in ROM, all variable data must be contained in data segments and cannot be initialized at run-time. Initializing values must reside in a code segment, and they must be copied down to their corresponding data segment locations at cold start. The assembly of these modules requires an assembler capable of supporting separate code and data segments; the segments must be assembled into REL file format. Use RMAC with 8080 source files.

The NETWRKIF module resides on the server and is identical to the NETWRKIF required to support CP/NET. See Section 4.3 for a discussion of NETWRKIF preparation.

A.4 Building the CP/NOS System

To generate a CP/NOS system ready for insertion into ROM, follow these steps:

1. Assemble the modules CPBIOS and CPNIOS.
2. Link the following modules together in the order shown, using LINK-80:

   CPNOS, CPNDOS, CPNIOS, CPBDOS, CPBIOS

   Locate the code segment where the ROM sits in the address space of the finished system. At least 1K (400 hexadecimal bytes) of RAM must be allocated for data segments. If the code segments are to be loaded into high memory (at F000H for a 4K system), data must be explicitly linked, using the D option, at least 1K in front of the code segments. For example,

   ```
   A>LINK CPNOS,CPNDOS,CPNIOS,CPBDOS,CPBIOS[LF000,DEC00]
   ```

These two steps produce an executable CP/NOS, capable of being programmed into ROM. At this stage, however, the system cannot be debugged from CP/M.

A.5 Debugging the System

You can create a version of CP/NOS that can be cold started from CP/M if a CP/M system with 64K RAM is available. First, type the following commands:

```
A>RMAC CPNIOS
A>RMAC CPBIOS
A>LINK CPNOS,CPNDOS,CPNIOS,CPBDOS,CPBIOS[LF000,DEC00]
A>GENHEX MVCPN0S 0100
A>GENHEX CPNOS 0200
A>PIP LDCPNOS.HEX=MVCPNOS.HEX[I],CPNOS.HEX[H]
A>LOAD LDCPNOS
```

This procedure produces a file LDCPNOS.COM that is directly executable from CP/M. LDCPNOS relocates the CPNOS module to location F000H and passes control to it, destroying CP/M and replacing it with CP/NOS.

Because CP/M is destroyed by this procedure, it is not advisable to run LDCPNOS under software debugger like DDT or SID, although you can run LDCPNOS under an in-circuit emulator. To run CP/NOS under DDT or SID, use the following procedure:

1. Link CPNOS so that all code and data reside below the address specified as END when the debugger is brought up:

   ```
   A>LINK CPNOS,CPNDOS,CPNIOS,CPBDOS,CPBIOS[L<org>,D<org-400H>]
   ```

   where <org> is the link origin.
2. `A>DDT CPNOS.COM`
3. Relocate CPNOS from location 100, where DDT loads it, to its link origin:

   ```
   -M100,<100+next-1>,<org>
   ```

   where next is the field specified by NEXT when the debugger loads CPNOS.COM, and <org> is the link origin.
4. Begin execution with appropriate diagnostics:

   ```
   -G<org>
   ```

   where <org> is the link origin.

Appendix B
CP/NET 1.2 Standard Message Formats

FMT	DID	SID	FNC	SIZ	MSG

- FMT = Message format code
- DID = Message destination processor ID
- SID = Message source processor ID
- FNC = MP/M function code
- SIZ = Data field length - 1
- MSG = Actual message, SIZ + 1 bytes long

Figure B-1. CP/NET 1.2 Logical Message Format

FMT CODE	FMT	DID	SID	FNC	siz	MSG	Comment
00	1	1	1	1	1	1-256	Preferred format
01	1	1	1	1	1	1-256	Returned result
02	1	1	1	1	2	1-65536	
03	1	1	1	1	2	1-65536	Returned result
04	1	2	2	1	1	1-256	
05	1	2	2	1	1	1-256	Returned result
06	1	2	2	1	2	1-65536	
07	1	2	2	1	2	1-65536	Returned result

Table B-1. Message Field Length Table

Appendix C
CP/NET 1.2 Logical Message Specifications

Messages for all CP/NET functions are defined in this appendix. These messages are logical messages. Any implementation of the SNIOS or NETWRKIF modules must always present messages to the NDOS or SERVER modules in the form presented here.

You must adhere to these formats when implementing a server that runs under an operating system other than MP/M II.

Notes:

- ss = Server ID
  rr = Requester ID
  xx = Don't care byte
  nn = Value specified
- All numeric values are in hexadecimal.
- All functions capable of returning extended errors are marked *EE*. Extended errors are assumed w
  bytes in length, with the following message format:

  SIZ = 01
  MSG(0) = FF
  MSG(1) = Extended Error Code

  Any response with SIZ = 01 is interpreted as an error, regardless of the value in MSG(0).
- Any message can return the *server not logged in* or *function not implemented on server* extended err

For functions that return with the user's FCB updated (messages that have an FCB in their response), the first byte of the FCB (drive designator) is never copied back from the response message. In some cases, the random record bytes are also not copied back.

For search functions, the entire directory entry (which is NOT an FCB) is copied back to the current DMA buffer, into the position indicated by the Directory Code result byte. This means that the DMA buffer is not the actual directory sector from the disk, but merely an accumulation of directory entries in an order determined by how they were found.

Functions that return the address of a system resource (Get Allocation Vector, Get DBP, Get Server Config), the data is kept in an NDOS buffer which is overwritten on subsequent calls. The user must copy data out as needed. The NDOS guarantees that the low byte of the address is never 0FFH, so that a valid address can be distinguished from an Extended Error Code.

For functions that return a drive vector (Get Login, Get R/O), there is no way to distingush between valid vectors and errors. The message format does not allow for errors, and the NDOS ignores the possibility of an error in the response. The actual vector returned to the user is a composite of data retrieved from all known servers and the local BDOS.

FMT	DID	SID	FNC	SIZ	MSG
0 SYSTEM RESET: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	00	00	• 00-00 = xx
01	rr	ss	00	01	• 00-00 = 0FFh • 01-01 = 00Ch

FMT	DID	SID	FNC	SIZ	MSG
1 CONSOLE INPUT: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	01	00	• 00-00 = xx
01	rr	ss	01	01	• 00-00 = 0FFh • 01-01 = 00Ch
2 CONSOLE OUTPUT: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	02	00	• 00-00 = xx
01	rr	ss	02	01	• 00-00 = 0FFh • 01-01 = 00Ch
3 RAW CONSOLE INPUT:					
00	ss	rr	03	00	• 00-00 = Server Console #
01	rr	ss	03	00	• 00-00 = Character Input
4 RAW CONSOLE OUTPUT:					
00	ss	rr	04	01	• 00-00 = Server Console # • 01-01 = Character to Output
01	rr	ss	04	00	• 00-00 = 00
5 LIST OUTPUT:					
00	ss	rr	05	nn	• 00-00 = Server List # • 01-nn = Characters to List Device (nn = 01 to 80)
01	rr	ss	05	00	• 00-00 = 00
6 DIRECT CONSOLE 1/0: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	06	00	• 00-00 = xx
01	rr	ss	06	01	• 00-00 = 0FFh • 01-01 = 00Ch
7 GET I/O BYTE: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	07	00	• 00-00 = xx
01	rr	ss	07	01	• 00-00 = 0FFh • 01-01 = 00Ch
8 SET 1/0 BYTE: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	08	00	• 00-00 = xx
01	rr	ss	08	01	• 00-00 = 0FFh • 01-01 = 00Ch
9 PRINT STRING: **NOT IMPLEMENTED AT SERVER**					

FMT	DID	SID	FNC	SIZ	MSG
00	ss	rr	09	00	• 00-00 = xx
01	rr	ss	09	01	• 00-00 = 0FFh • 01-01 = 00Ch
10 READ CONSOLE BUFFER: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	0A	00	• 00-00 = xx
01	rr	ss	0A	01	• 00-00 = 0FFh • 01-01 = 00Ch
11 GET CONSOLE STATUS:					
00	ss	rr	0B	00	• 00-00 = Server Console #
01	rr	ss	0B	00	• 00-00 = Console Status Byte
12 RETURN VERSION NUMBER: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	0C	00	• 00-00 = xx
01	rr	ss	0C	01	• 00-00 = 0FFh • 01-01 = 00Ch
13 RESET DISK SYSTEM: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	0D	00	• 00-00 = xx
01	rr	ss	0D	01	• 00-00 = 0FFh • 01-01 = 00Ch
14 SELECT DISK: *EE*					
00	ss	rr	0E	00	• 00-00 = Selected Disk
01	rr	ss	0E	00	• 00-00 = Return Code
15 OPEN FILE: *EE*					
00	ss	rr	0F	2C	• 00-00 = User Number • 01-24 = FCB • 25-2C = Password
01	rr	ss	0F	24	• 00-00 = Directory Code • 01-24 = FCB 01 not copied to user; 22-24 (file ID) not copied to user unless F5' set and not F6'.
16 CLOSE FILE: *EE*					
00	ss	rr	10	2C	• 00-00 = User Number • 01-24 = FCB • 25-2C = Password (ignored)
01	rr	ss	10	24	• 00-00 = Directory Code • 01-24 = FCB 01 not copied to user; 22-24 (file ID) not copied to user unless F5' set and not F6'.

<table>
<tr><th>FMT</th><th>DID</th><th>SID</th><th>FNC</th><th>SIZ</th><th>MSG</th></tr>
<tr><td colspan="6">17 SEARCH FOR FIRST: *EE*</td></tr>
<tr><td>00</td><td>ss</td><td>rr</td><td>11</td><td>25</td><td>00-00 = Current Disk if FCB(0)='?'
01-01 = User Number
02-25 = FCB</td></tr>
<tr><td rowspan="2">01</td><td rowspan="2">rr</td><td rowspan="2">ss</td><td rowspan="2">11</td><td>20</td><td>00-00 = Directory Code
01-20 = Directory Entry
Directory Entry copied to current DMA buffer
based on Directory Code.</td></tr>
<tr><td>80/0</td><td>00-00 = Directory Code
01-80 = Directory Sector
00-00 = Directory Code
Extensions for "full search" mode, incl. CP/M 3.</td></tr>
<tr><td colspan="6">18 SEARCH FOR NEXT: *EE*</td></tr>
<tr><td>00</td><td>ss</td><td>rr</td><td>12</td><td>01</td><td>00-00 = xx
01-01 = User Number</td></tr>
<tr><td rowspan="2">01</td><td rowspan="2">rr</td><td rowspan="2">ss</td><td rowspan="2">12</td><td>20</td><td>00-00 = Directory Code
01-20 = Directory Entry
Directory Entry copied to current DMA buffer
based on Directory Code.</td></tr>
<tr><td>80/0</td><td>00-00 = Directory Code
01-80 = Directory Sector
00-00 = Directory Code
Extensions for "full search" mode, incl. CP/M 3.</td></tr>
<tr><td colspan="6">19 DELETE FILE: *EE*</td></tr>
<tr><td>00</td><td>ss</td><td>rr</td><td>13</td><td>24</td><td>00-00 = User Number
01-24 = FCB</td></tr>
<tr><td>01</td><td>rr</td><td>ss</td><td>13</td><td>00</td><td>00-00 = Directory Code</td></tr>
<tr><td colspan="6">20 READ SEQUENTIAL: *EE*</td></tr>
<tr><td>00</td><td>ss</td><td>rr</td><td>14</td><td>24</td><td>00-00 = User Number
01-24 = FCB</td></tr>
<tr><td>01</td><td>rr</td><td>ss</td><td>14</td><td>A4</td><td>00-00 = Return Code
01-24 = FCB
25-A4 = Sector of Data Read
01 and 22-24 not copied to user.</td></tr>
<tr><td colspan="6">21 WRITE SEQUENTIAL: *EE*</td></tr>
<tr><td>00</td><td>ss</td><td>rr</td><td>15</td><td>A4</td><td>00-00 = User Number
01-24 = FCB
25-A4 = Sector of Data to Write</td></tr>
<tr><td>01</td><td>rr</td><td>ss</td><td>15</td><td>24</td><td>00-00 = Return Code
01-24 = FCB
01 and 22-24 not copied to user.</td></tr>
<tr><td colspan="6">22 MAKE FILE: *EE*</td></tr>
</table>

FMT	DID	SID	FNC	SIZ	MSG
00	ss	rr	16	24	• 00-00 = User Number • 01-24 = FCB
01	rr	ss	16	24	• 00-00 = Directory Code • 01-24 = FCB 01 not copied to user; 22-24 (file ID) not copied to user unless F5' set and not F6'.
23 RENAME FILE: *EE*					
00	ss	rr	17	24	• 00-00 = User Number • 01-24 = FCB in RENAME format
01	rr	ss	17	00	• 00-00 = Directory Code
24 RETURN LOGIN VECTOR: Message Sent for Each Remote Drive, Results combined with local BDOS					
00	ss	rr	18	00	• 00-00 = xx
01	rr	ss	18	01	• 00-01 = Login Vector
25 RETURN CURRENT DISK: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	19	00	• 00-00 = xx
01	rr	ss	19	01	• 00-00 = 0FFh • 01-01 = 00Ch
26 SET DMA ADDRESS: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	1A	00	• 00-00 = xx
01	rr	ss	1A	01	• 00-00 = 0FFh • 01-01 = 00Ch
27 GET ALLOCATION VECTOR ADDRESS: *EE*					
00	ss	rr	1B	00	• 00-00 = Current Disk
01	rr	ss	1B	02-FF	• 00-FF = Allocation Vector NDOS guarantees low byte of returned address cannot be FF.
28 WRITE PROTECT DISK: *EE*					
00	ss	rr	1C	00	• 00-00 = Current Disk
01	rr	ss	1C	00	• 00-00 = 00
29 GET R/O VECTOR: Message Sent for Each Remote Drive, Results combined with local BDOS					
00	ss	rr	1D	00	• 00-00 = xx
01	rr	ss	1D	01	• 00-01 = R/O Vector
30 SET FILE ATTRIBUTES: *EE*					
00	ss	rr	1E	24	• 00-00 = User Number • 01-24 = FCB with File Attributes Set

FMT	DID	SID	FNC	SIZ	MSG
01	rr	ss	1E	00	• 00-00 = Directory Code
31 GET DISK PARAMETER ADDRESS: *EE*					
00	ss	rr	1F	00	• 00-00 = Current Disk
01	rr	ss	1F	0F	• 00-0F = Disk Parameter Block
32 SET/GET USER CODE: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	20	00	• 00-00 = xx
01	rr	ss	20	01	• 00-00 = 0FFh • 01-01 = 00Ch
33 READ RANDOM: *EE*					
00	ss	rr	21	24	• 00-00 = User Number • 01-24 = FCB
01	rr	ss	21	A4	• 00-00 = Return Code • 01-24 = FCB • 25-A4 = Sector of Data Read 01 not copied to user.
34 WRITE RANDOM: *EE*					
00	ss	rr	22	A4	• 00-00 = User Number • 01-24 = FCB • 25-A4 = Sector of Data to Write
01	rr	ss	22	24	• 00-00 = Return Code • 01-24 = FCB 01 not copied to user.
35 COMPUTE FILE SIZE: *EE*					
00	ss	rr	23	24	• 00-00 = User Number • 01-24 = FCB
01	rr	ss	23	24	• 00-00 = Return Code • 01-24 = FCB
36 SET RANDOM RECORD: *EE*					
00	ss	rr	24	24	• 00-00 = User Number • 01-24 = FCB
01	rr	ss	24	24	• 00-00 = Return Code • 01-24 = FCB 01 not copied to user.
37 RESET DRIVE: *EE* Message Sent to All Affected Servers					
00	ss	rr	25	01	• 00-01 = Drive Vector
01	rr	ss	25	00	• 00-00 = Return Code
38 ACCESS DRIVE: *EE* Message Sent to All Affected Servers					

FMT	DID	SID	FNC	SIZ	MSG
00	ss	rr	26	01	• 00-01 = Drive Vector
01	rr	ss	26	00	• 00-00 = Return Code
39 FREE DRIVE: *EE* Message Sent to All Affected Servers					
00	ss	rr	27	01	• 00-01 = Drive Vector
01	rr	ss	27	00	• 00-00 = Return Code
40 WRITE RANDOM WITH ZERO FILL: *EE*					
00	ss	rr	28	A4	• 00-00 = User Number • 01-24 = FCB • 25-A4 = Sector of Data to Write
01	rr	ss	28	24	• 00-00 = Return Code • 01-24 = FCB 01 not copied to user.
42 LOCK RECORD: *EE*					
00	ss	rr	2A	26	• 00-00 = User Number • 01-24 = FCB • 25-26 = File ID
01	rr	ss	2A	24	• 00-00 = Return Code • 01-24 = FCB 01 not copied to user.
43 UNLOCK RECORD: *EE*					
00	ss	rr	2B	26	• 00-00 = User Number • 01-24 = FCB • 25-26 = File ID
01	rr	ss	2B	24	• 00-00 = Return Code • 01-24 = FCB 01 not copied to user.
45 SET BDOS ERROR MODE: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	2D	00	• 00-00 = xx
01	rr	ss	2D	01	• 00-00 = 0FFh • 01-01 = 00Ch
46 GET DISK FREE SPACE: Extension for CP/M 3.					
00	ss	rr	2E	00	• 00-00 = Drive
01	rr	ss	2E	02	• 00-02 = Free Space, little-endian Copied to current DMA buffer.
48 FLUSH BUFFERS: Extension for CP/M 3.					
00	ss	rr	30	00	• 00-00 = Purge Code

FMT	DID	SID	FNC	SIZ	MSG
01	rr	ss	30	00	• 00-00 = Return Code
64 LOGIN: *EE*					
00	ss	rr	40	07	• 00-07 = Password, 8 ASCII Chars
01	rr	ss	40	00	• 00-00 = Return Code
65 LOGOFF: *EE*					
00	ss	rr	41	00	• 00-00 = xx
01	rr	ss	41	00	• 00-00 = Return Code
66 SEND MESSAGE ON NETWORK: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	42	xx	• 00-FF = xx
01	rr	ss	42	01	• 00-00 = 0FFh • 01-01 = 00Ch
67 RECEIVE MESSAGE ON NETWORK: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	43	00	• 00-00 = xx
01	rr	ss	43	01	• 00-00 = 0FFh • 01-01 = 00Ch
68 GET NETWORK STATUS: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	44	00	• 00-00 = xx
01	rr	ss	44	01	• 00-00 = 0FFh • 01-01 = 00Ch
69 GET CONFIGURATION TABLE ADDRESS: **NOT IMPLEMENTED AT SERVER**					
00	ss	rr	45	00	• 00-00 = xx
01	rr	ss	45	01	• 00-00 = 0FFh • 01-01 = 00Ch
70 SET COMPATIBILITY ATTRIBUTES: Message Sent to All Known Servers					
00	ss	rr	46	00	• 00-00 = Compatibility Attributes
01	rr	ss	46	00	• 00-00 = xx
71 RETURN SERVER CONFIGURATION: *EE*					
00	ss	rr	47	00	• 00-00 = xx
01	rr	ss	47	16	• 00-00 = Server Temporary File Drive • 01-01 = Server Status Byte • 02-02 = Server ID • 03-03 = Maximum Number of Requesters • 04-04 = Number Logged In • 05-06 = Login Vector

FMT	DID	SID	FNC	SIZ	MSG
					• 07-16 = Requester ID's
98 FREE BLOCKS: Extension for CP/M 3.					
00	ss	rr	62	00	• 00-00 = xx
01	rr	ss	62	00	• 00-00 = Return Code
99 TRUNCATE FILE: *EE* Extension for CP/M 3.					
00	ss	rr	63	24	• 00-00 = User Number • 01-24 = FCB
01	rr	ss	63	24	• 00-00 = Directory Code • 01-24 = FCB
101 GET DIRECTORY LABEL BYTE: Extension for CP/M 3.					
00	ss	rr	65	00	• 00-00 = Drive
01	rr	ss	65	00	• 00-00 = Dir Mode Byte
102 GET FILE TIME STAMPS: *EE* Extension for CP/M 3.					
00	ss	rr	66	24	• 00-00 = User Number • 01-24 = FCB
01	rr	ss	66	24	• 00-00 = Directory code • 01-24 = FCB with timestamps in d8-d15
105 GET DATE AND TIME: Extension for CP/M 3. Not issued by NDOS.					
00	ss	rr	69	00	• 00-00 = xx
01	rr	ss	69	04	• 00-01 = Date, days since 12/31/1977 • 02 = Hours (BCD) • 03 = Minutes (BCD) • 04 = Seconds (BCD)
106 SET DEFAULT PASSWORD: Message Sent to All Known Servers					
00	ss	rr	6A	07	• 00-07 = Default Password to be Set
01	rr	ss	6A	00	• 00-00 = Return Code

Table C-1. Conventional CP/NET Messages

Appendix D
NDOS Function Summary

Code	Function Name	Input Parameters	Output Results
38	Access Drive	DE = Drive Vector	none
39	Free Drive	DE = Drive Vector	none
42	Lock Record	DE = FCB Address	A = Err Code
43	Unlock Record	DE = FCB Address	A = Err Code
45	Set BDOS Error Mode	E = Error Mode	none
64	Login	see definition	A = Err Code
65	Logoff	E = Server ID	none
66	Send Message on Ntwrk	DE = Message Adr	A = Err Code
67	Receive Msg from Ntwk	DE = Message Adr	A = Err Code
68	Get Network Status	none	A = Status byte
69	Get Config Table Adr	none	HL = Table Adr
70	Set Compat. Attrs.	E = attributes	none
71	Get Server Config.	E = Server ID	HL= Table Adr
106	Set Default Password	see definition	none

Table D-1. NDOS Functions

Appendix E
A Simple RS-232C CP/NET System

Digital Research developed a relatively simple RS-232C point to-point protocol to provide a demonstration vehicle for CP/NET and to encourage compatibility among hardware vendors. The protocol, as implemented in the sample SNIOS and NETWRKIF in this appendix, breaks the logical message into a fixed header and a variable length data portion the size of which is obtained from the fixed header. This simplifies operation with DMA channels that need terminal counts and also provides a checksum for the header that contains the SIZ field.

This protocol can be implemented between any requester and server that support an extra RS-232 console port.

E.1 Protocol Handshake

The protocol handshake is detailed in Figure E-1.

```
 Source                Destination   Comment

5 = ENQ  ----------->

         <-----------  6 = ACK

1 = SOH  ----------->
    FMT  ----------->
    DID  ----------->
    SID  ----------->
    FNC  ----------->
    SIZ  ----------->
    HCS  ----------->                Modulo 256 sum from
                                       SOH to HCS = 0
         <-----------  6 = ACK

2 = STX  ----------->
    DB0  ----------->                First data byte

    DBn  ----------->
3 = ETX  ----------->
    CKS  ----------->                Modulo 256 sum from
4 = EOT  ----------->                  STX to CKS = 0
         <-----------  6 = ACK
```

Figure E-1. Protocol Handshake

E.2 Binary Protocol Message Format

Data integrity for this protocol is maintained by a simple checksum, shown in Figure E-2, on both the header and the actual message.

Figure E-2. Binary Protocol Message Format

Message format codes 00 & 01 are recommended.

Field Description:

- ENQ = Enquire, one byte, 05H.
- SOH = Start of Header, one byte, 01H.
- FMT,DID,SID,FNC,SIZ = as defined in Appendix A, one byte per field.
- HCS = Header Checksum, one byte. This is a simple horizontal checksum, computed by adding together all the bytes of the message, starting with the SOH, to the SIZ byte of the header field modulo 256, complementing the result, and adding one. The entire message, from the SOH to and including the HCS, should add up to zero.
- STX = Start of Data, one byte, 02H.
- MSG = SIZ + 1 byte long.
- ETX = End of Data, one byte, 03H.
- CKS = Checksum, one byte. This is a simple horizontal checksum, computed by adding together all the bytes of the message, starting with the STX, to the last byte of the MSG field modulo 256, complementing the result, and adding one. The entire message, from the STX to and including the CKS, should add up to zero.
- EOT = End of Transmission, one byte, 04H.

E.3 ASCII Protocol Message Format

If the RS-232 link is not capable of transmitting 8-bit binary data, you might have to transmit each nibble of the message as a 7 bit ASCII character.

Note: the 7-bit ASCII network protocol is identical to the 8-bit protocol except that it requires twice as many bytes because each byte is transmitted in hexadecimal ASCII format.

The ASCII network protocol message format is detailed in Figure E-3.

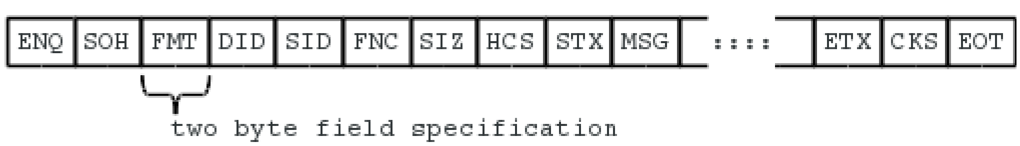


Figure E-3. ASCII Protocol Message Format

Message format codes 00 & 01 are recommended.

Field Description:

- ENQ = Enquire, one byte, 05H.
- SOH = Start of Header, one byte, 01H.
- FMT,DID,SID,FNC,SIZ = as defined in Appendix A, two bytes per field.
- HCS = Header Checksum, 2 bytes (Hex-ASCII) . This is a simple horizontal checksum. It is computed by adding together all the bytes of the message, starting with the SOH, to the SIZ of the header field modulo 256, complementing the result, and adding one. The entire message, from the

SOH to the including the HCS, should add up to zero.

- STX = Start of Data, one byte, 02H.
- MSG = 2 * (SIZ + 1) bytes long.
- ETX = End of Data, one byte, 03H.
- CKS =Checksum, two bytes (Hex-ASCII) . This is a simple horizontal checksum. It is computed by adding together all the bytes of the message, starting with the STX, to the last byte of the MSG field modulo 256, complementing the result and adding one. The entire message, from the FMT to and including the CKS, should add up to zero.
- EOT = End of Transmission, one byte, 04H.

E.4 Modifying the SNIOS

The sample SNIOS can be modified for almost any requester that has a spare console port. To do so, follow these steps:

1. Obtain assembled listings of the SNIOS.ASM source file that require modification. You can use MAC, RMAC, or ASM. if you use ASM, the title, name, if, and else statements must be removed from the source files to assemble correctly. Using RMAC is highly recommended because it simplifies the task of generating the SPR files when used in conjunction with LINK. Otherwise, the SPR files must be generated in the same manner as for MP/M II XIOS.SPR generation.

```
A>RMAC SNIOS
```

2. Study the SNIOS.PRN listing. Notice the ASCII equate. If true, it specifies that the message format is 7-bit ASCII. If false, it specifies a binary 8-bit message format. The ASCII mode is sometimes useful in debugging, but in practice do not use it where it is possible to transmit 8 bit serial data.

The only code that requires modification in the SNIOS.ASM file is contained in the CHAROUT, CHARIN, and DELAY procedures. The CHAROUT and CHARIN procedures can be conditionally assembled for a Dynabyte DB8/2, now called DB8/5200, a Digital Microsystems DSC-2 or an ALTOS 8000-2. The NOPs in the CHAROUT procedure are simply padding, so the length of the DB8/2 SNIOS and DSC-2 SNIOS is the same, which helps in the debugging of these two versions.

Perhaps the most critical area in the SNIOS that requires adjustment for a specific network configuration is in the timeout code of the CHARIN procedure. If too little time is allowed, the server might not be able to complete the function because of a heavy request load from the requesters. If too much time is specified, communication breaks on the network can go undetected for a period of time, making both error recovery and precise detection difficult. Note that this is a logical timeout, not a data-link timeout. The logical timeout determines how long the requester expects the server to take between the time it receives the message and the time it returns a response message.

Another critical parameter that requires adjustment for different environments is ALWAYS$RETRY. This equate, when true, controls conditional assembly that always produces retries on network failures. In this mode of operation, it is possible to recover from broken communication between the requester and a server. However, ALWAYS$RETRY does hang the requester in a busy retry mode when failures occur.

```
                                title   'Requester Network I/O System for CP/NET 1.2'
                                page    54

                        ;***************************************************************
                        ;***************************************************************
                        ;**                                                           **
                        ;**  R e q u e s t e r   N e t w o r k   I / O   S y s t e m  **
                        ;**                                                           **
                        ;***************************************************************
                        ;***************************************************************

                        ;/*
                        ;  Copyright (C) 1980, 1981, 1982
                        ;  Digital Research
                        ;  P.O. Box 579
                        ;  Pacific Grove, CA 93950
                        ;
                        ;  Revised:  October 5, 1982
                        ;*/

        0000 =          false   equ     0
        FFFF =          true    equ     not false

        0000 =          cpnos   equ     false           ; cp/net system

        0000 =          DSC2    equ     false
        0000 =          DB82    equ     false
        FFFF =          Altos   equ     true

        FFFF =          always$retry    equ     true    ; force continuous retries

        0000 =          modem   equ     false

        0000 =          ASCII   equ     false

        0000 =          debug   equ     false

                                CSEG
                                if      cpnos
                                extrn   BDOS
                                else
        0005 =          BDOS    equ     0005h
                                endif

                        NIOS:
                                public  NIOS
                        ;       Jump vector for SNIOS entry points
        0000 C3A900             jmp     ntwrkinit       ; network initialization
        0003 C3B800             jmp     ntwrksts        ; network status
        0006 C3C300             jmp     cnfgtbladr      ; return config table addr
        0009 C3C700             jmp     sendmsg         ; send message on network
        000C C33301             jmp     receivemsg      ; receive message from network
        000F C3DD01             jmp     ntwrkerror      ; network error
        0012 C3DE01             jmp     ntwrkwboot      ; network warm boot
```

```
                        if      DB82
               slave$ID         equ     12h     ; slave processor ID number
                        endif
                        if      DSC2
               slave$ID         equ     34h
                        endif
                        if      Altos
0056 =         slave$ID         equ     56h
                        endif

                        if      cpnos
               ;        Initial Slave Configuration Table
               Initconfigtbl:
                        db      0000$0000b      ; network status byte
                        db      slave$ID        ; slave processor ID number
                        db      84h,0           ; A:  Disk device
                        db      81h,0           ; B:   "
                        db      82h,0           ; C:   "
                        db      83h,0           ; D:   "
                        db      80h,0           ; E:   "
                        db      85h,0           ; F:   "
                        db      86h,0           ; G:   "
                        db      87h,0           ; H:   "
                        db      88h,0           ; I:   "
                        db      89h,0           ; J:   "
                        db      8ah,0           ; K:   "
                        db      8bh,0           ; L:   "
                        db      8ch,0           ; M:   "
                        db      8dh,0           ; N:   "
                        db      8eh,0           ; O:   "
                        db      8fh,0           ; P:   "
                        db      0,0             ; console device
                        db      0,0             ; list device:
                        db      0               ;       buffer index
                        db      0               ;       FMT
                        db      0               ;       DID
                        db      slave$ID        ;       SID
                        db      5               ;       FNC
               initcfglen equ  $-initconfigtbl
                        endif

0000 =         defaultmaster   equ     00h

               wboot$msg:                       ; data for warm boot routine
0015 3C5761726D         db      ''
0020 24                 db      '$'

               networkerrmsg:
0021 4E6574776F         db      'Network Error'
002E 24                 db      '$'


                        page
```

```

                                DSEG


                        ;       Slave Configuration Table
                        configtbl:

                        Network$status:
         0000                   ds      1               ; network status byte
         0001                   ds      1               ; slave processor ID number
         0002                   ds      2               ; A:  Disk device
         0004                   ds      2               ; B:   "
         0006                   ds      2               ; C:   "
         0008                   ds      2               ; D:   "
         000A                   ds      2               ; E:   "
         000C                   ds      2               ; F:   "
         000E                   ds      2               ; G:   "
         0010                   ds      2               ; H:   "
         0012                   ds      2               ; I:   "
         0014                   ds      2               ; J:   "
         0016                   ds      2               ; K:   "
         0018                   ds      2               ; L:   "
         001A                   ds      2               ; M:   "
         001C                   ds      2               ; N:   "
         001E                   ds      2               ; O:   "
         0020                   ds      2               ; P:   "

         0022                   ds      2               ; console device

         0024                   ds      2               ; list device:
         0026                   ds      1               ;       buffer index
         0027 00                db      0               ;       FMT
         0028 00                db      0               ;       DID
         0029 56                db      Slave$ID        ;       SID (CP/NOS must still initialize)
         002A 05                db      5               ;       FNC
         002B                   ds      1               ;       SIZ
         002C                   ds      1               ;       MSG(0)  List number
         002D                   ds      128             ;       MSG(1) ... MSG(128)

                        msg$adr:
         00AD                   ds      2               ; message address
                                if      modem
                        timeout$retries equ 0           ; timeout a max of 256 times
                                else
         0064 =         timeout$retries equ 100         ; timeout a max of 100 times
                                endif
         000A =         max$retries equ 10              ; send message max of 10 times
                        retry$count:
         00AF                   ds      1

                        FirstPass:
         00B0 FF                db      0ffh

                        ;       Network Status Byte Equates
```

```
                ;
0010 =          active          equ     0001$0000b      ; slave logged in on network
0002 =          rcverr          equ     0000$0010b      ; error in received message
0001 =          senderr         equ     0000$0001b      ; unable to send message

                ;       General Equates
                ;
0001 =          SOH     equ     01h             ; Start of Header
0002 =          STX     equ     02h             ; Start of Data
0003 =          ETX     equ     03h             ; End of Data
0004 =          EOT     equ     04h             ; End of Transmission
0005 =          ENQ     equ     05h             ; Enquire
0006 =          ACK     equ     06h             ; Acknowledge
000A =          LF      equ     0ah             ; Line Feed
000D =          CR      equ     0dh             ; Carriage Return
0015 =          NAK     equ     15h             ; Negative Acknowledge

0002 =          conout  equ     2               ; console output function
0009 =          print   equ     9               ; print string function
0043 =          rcvmsg  equ     67              ; receive message NDOS function
0040 =          login   equ     64              ; Login NDOS function

                ;       I/O Equates
                ;
                        if      DB82
                stati   equ     83h
                mski    equ     08h
                dprti   equ     80h

                stato   equ     83h
                msko    equ     10h
                statc   equ     81h
                mskc    equ     20h
                dprto   equ     86h
                        endif

                        if      DSC2
                        if      modem
                stati   equ     59h
                mski    equ     02h
                dprti   equ     58h

                stato   equ     59h
                msko    equ     01h
                dprto   equ     58h
                        else
                stati   equ     51h
                mski    equ     02h
                dprti   equ     50h

                stato   equ     51h
                msko    equ     01h
                dprto   equ     50h
                        endif
```

```
                        endif

                        if      Altos
001F =          stati   equ     1fh
0001 =          mski    equ     01h
001E =          dprti   equ     1eh

001F =          stato   equ     1fh
0004 =          msko    equ     04h
001E =          dprto   equ     1eh
                        endif



                        page
```

```

                        CSEG
                ;       Utility Procedures
                ;
                delay:                          ; delay for c[a] * 0.5 milliseconds
 002F 3E06              mvi     a,6
                delay1:
 0031 0E86              mvi     c,86h
                delay2:
 0033 0D                dcr     c
 0034 C23300            jnz     delay2
 0037 3D                dcr     a
 0038 C23100            jnz     delay1
 003B C9                ret

                        if      ASCII
                Nib$out:                        ; A = nibble to be transmitted in ASCII
                        cpi     10
                        jnc     nibAtoF         ; jump if A-F
                        adi     '0'
                        mov     c,a
                        jmp     Char$out
                nibAtoF:
                        adi     'A'-10
                        mov     c,a
                        jmp     Char$out
                        endif

                Pre$Char$out:
 003C 7A                mov     a,d
 003D 81                add     c
 003E 57                mov     d,a             ; update the checksum in D

                nChar$out:                      ; C = byte to be transmitted
                        if      Altos
 003F 3E10              mvi     a,10h
 0041 D31F              out     stato
                        endif
 0043 DB1F              in      stato
 0045 E604              ani     msko
 0047 CA3F00            jz      nChar$out

                        if      DB82
                        in      statc
                        ani     mskc
                        jz      nChar$out
                        endif

                        if      DSC2
                        nop                     ; these NOP's make DB8/2 & DSC2
                        nop                     ;  versions the same length - saves
                        nop                     ;  a second listing
                        nop
                        nop
```

```
                                nop
                                nop
                                endif

        004A 79                 mov     a,c
        004B D31E               out     dprto
        004D C9                 ret
                        ;
                        Char$out:
        004E CD3F00             call    nChar$out
                                if      Altos
        0051 E3E3E3E3           xthl! xthl! xthl! xthl
        0055 E3E3E3E3           xthl! xthl! xthl! xthl
        0059 E3E3E3E3           xthl! xthl! xthl! xthl  ;delay 54 usec
        005D C9                 ret
                                else
                                jmp     delay           ; delay after each Char sent to Mstr
                        ;       ret
                                endif

                                if      ASCII
                        Nib$in:                         ; return nibble in A register
                                call    Char$in
                                rc
                                ani     7fh
                                sui     '0'
                                cpi     10
                                jc      Nib$in$rtn      ; must be 0-9
                                adi     ('0'-'A'+10) and 0ffh
                                cpi     16
                                jc      Nib$in$rtn      ; must be 10-15
                                lda     network$status
                                ori     rcverr
                                sta     network$status
                                mvi     a,0
                                stc                     ; carry set indicating err cond
                                ret

                        Nib$in$rtn:
                                ora     a               ; clear carry & return
                                ret
                                endif

                        xChar$in:
        005E 0664               mvi     b,100           ; 100 ms  corresponds to longest possible
        0060 C36500             jmp     char$in0        ;wait between master operations

                        Char$in:                        ; return byte in A register
                                                        ;  carry set on rtn if timeout
                                if      modem
                                mvi     b,0             ; 256 ms = 7.76 chars @ 300 baud
                                else
                                if      Altos
        0063 0603               mvi     b,3             ; 3 ms = 50 chars @ 125k baud
```

```
                                else
                                mvi     b,50            ; 50 ms = 50 chars @ 9600 baud
                                endif
                                endif
                        Char$in0:
        0065 0E5A               mvi     c,5ah
                        Char$in1:
                                if      Altos
        0067 3E00               mvi     a,0
        0069 D31F               out     stati
                                endif
        006B DB1F               in      stati
        006D E601               ani     mski
        006F C27C00             jnz     Char$in2
        0072 0D                 dcr     c
        0073 C26700             jnz     Char$in1
        0076 05                 dcr     b
        0077 C26500             jnz     Char$in0
        007A 37                 stc                     ; carry set for err cond = timeout
        007B C9                 ret
                        Char$in2:
        007C DB1E               in      dprti
        007E C9                 ret                     ; rtn with raw char and carry cleared

                        Net$out:                        ; C = byte to be transmitted
                                                        ; D = checksum
        007F 7A                 mov     a,d
        0080 81                 add     c
        0081 57                 mov     d,a

                                if      ASCII
                                mov     a,c
                                mov     b,a
                                rar
                                rar
                                rar
                                rar
                                ani     0FH             ; mask HI-LO nibble to LO nibble
                                call    Nib$out
                                mov     a,b
                                ani     0FH
                                jmp     Nib$out

                                else
        0082 C34E00             jmp     Char$out
                                endif

                        Msg$in:                         ; HL = destination address
                                                        ; E  = # bytes to input
        0085 CD9000             call    Net$in
        0088 D8                 rc
        0089 77                 mov     m,a
        008A 23                 inx     h
        008B 1D                 dcr     e
```

```
008C C28500             jnz     Msg$in
008F C9                 ret

                Net$in:                         ; byte returned in A register
                                                ; D  = checksum accumulator

                        if      ASCII
                        call    Nib$in
                        rc
                        add     a
                        add     a
                        add     a
                        add     a
                        push    psw
                        call    Nib$in
                        pop     b
                        rc
                        ora     b

                        else
0090 CD6300             call    Char$in         ;receive byte in Binary mode
0093 D8                 rc
                        endif

                chks$in:
0094 47                 mov     b,a
0095 82                 add     d               ; add & update checksum accum.
0096 57                 mov     d,a
0097 B7                 ora     a               ; set cond code from checksum
0098 78                 mov     a,b
0099 C9                 ret

                Msg$out:                        ; HL = source address
                                                ; E  = # bytes to output
                                                ; D  = checksum
                                                ; C  = preamble byte
009A 1600               mvi     d,0             ; initialize the checksum
009C CD3C00             call    Pre$Char$out    ; send the preamble character
                Msg$out$loop:
009F 4E                 mov     c,m
00A0 23                 inx     h
00A1 CD7F00             call    Net$out
00A4 1D                 dcr     e
00A5 C29F00             jnz     Msg$out$loop
00A8 C9                 ret

                        page
```

```

                        ;       Network Initialization
                        ntwrkinit:

                                if      cpnos           ; copy down network assignments
                                lxi     h,Initconfigtbl
                                lxi     d,configtbl
                                mvi     c,initcfglen
                        initloop:
                                mov     a,m
                                stax    d
                                inx     h
                                inx     d
                                dcr     c
                                jnz     initloop                ; initialize config tbl from ROM

                                else
        00A9 3E56               mvi     a,slave$ID              ;initialize slave ID byte
        00AB 320100             sta     configtbl+1             ;  in the configuration tablee
                                endif

                        ;       device initialization, as required

                                if      Altos
        00AE 3E47               mvi     a,047h
        00B0 D30E               out     0eh
        00B2 3E01               mvi     a,1
        00B4 D30E               out     0eh
                                endif

                                if      DSC2 and modem
                                mvi     a,0ceh
                                out     stato
                                mvi     a,027h
                                out     stato
                                endif

                                if      cpnos
                                call    loginpr                 ; login to a master
                                endif

                        initok:
        00B6 AF                 xra     a                       ; return code is 0=success
        00B7 C9                 ret


                                page
```

```

                  ;       Network Status
                  ntwrksts:
00B8 3A0000               lda     network$status
00BB 47                   mov     b,a
00BC E6FC                 ani     not (rcverr+senderr)
00BE 320000               sta     network$status
00C1 78                   mov     a,b
00C2 C9                   ret



                  ;       Return Configuration Table Address
                  cnfgtbladr:
00C3 210000               lxi     h,configtbl
00C6 C9                   ret


                          page
```

```

                        ;       Send Message on Network
                        sendmsg:                        ; BC = message addr
00C7 60                         mov     h,b
00C8 69                         mov     l,c             ; HL = message address
00C9 22AD00                     shld    msg$adr
                        re$sendmsg:
00CC 3E0A                       mvi     a,max$retries
00CE 32AF00                     sta     retry$count     ; initialize retry count
                        send:
00D1 2AAD00                     lhld    msg$adr
00D4 0E05                       mvi     c,ENQ
00D6 CD4E00                     call    Char$out        ; send ENQ to master
00D9 1664                       mvi     d,timeout$retries
                        ENQ$response:
00DB CD6300                     call    Char$in
00DE D2E800                     jnc     got$ENQ$response
00E1 15                         dcr     d
00E2 C2DB00                     jnz     ENQ$response
00E5 C32B01                     jmp     Char$in$timeout
                        got$ENQ$response:
00E8 CD1E01                     call    get$ACK0
00EB 0E01                       mvi     c,SOH
00ED 1E05                       mvi     e,5
00EF CD9A00                     call    Msg$out         ; send SOH FMT DID SID FNC SIZ
00F2 AF                         xra     a
00F3 92                         sub     d
00F4 4F                         mov     c,a
00F5 CD7F00                     call    net$out         ; send HCS (header checksum)
00F8 CD1801                     call    get$ACK
00FB 2B                         dcx     h
00FC 5E                         mov     e,m
00FD 23                         inx     h
00FE 1C                         inr     e
00FF 0E02                       mvi     c,STX
0101 CD9A00                     call    Msg$out         ; send STX DB0 DB1 ...
0104 0E03                       mvi     c,ETX
0106 CD3C00                     call    Pre$Char$out    ; send ETX
0109 AF                         xra     a
010A 92                         sub     d
010B 4F                         mov     c,a
010C CD7F00                     call    Net$out         ; send the checksum
010F 0E04                       mvi     c,EOT
0111 CD3F00                     call    nChar$out       ; send EOT
0114 CD1801                     call    get$ACK         ; (leave these
0117 C9                         ret                     ;               two instructions)

                        get$ACK:
0118 CD6300                     call    Char$in
011B DA2301                     jc      send$retry      ; jump if timeout
                        get$ACK0:
011E E67F                       ani     7fh
0120 D606                       sui     ACK
0122 C8                         rz
```

```
                        send$retry:
        0123 E1                 pop     h               ; discard return address
        0124 21AF00             lxi     h,retry$count
        0127 35                 dcr     m
        0128 C2D100             jnz     send            ; send again unles max retries
                        Char$in$timeout:
        012B 3E01               mvi     a,senderr

                                if      always$retry
        012D CDD201             call    error$return
        0130 C3CC00             jmp     re$sendmsg
                                else
                                jmp     error$return
                                endif

                                page
```

```

                        ;       Receive Message from Network
                        receivemsg:                     ; BC = message addr
        0133 60                 mov     h,b
        0134 69                 mov     l,c             ; HL = message address
        0135 22AD00             shld    msg$adr
                        re$receivemsg:
        0138 3E0A               mvi     a,max$retries
        013A 32AF00             sta     retry$count     ; initialize retry count
                        re$call:
        013D CD4F01             call    receive         ; rtn from receive is receive error

                        receive$retry:
        0140 21AF00             lxi     h,retry$count
        0143 35                 dcr     m
        0144 C23D01             jnz     re$call
                        receive$timeout:
        0147 3E02               mvi     a,rcverr

                                if      always$retry
        0149 CDD201             call    error$return
        014C C33801             jmp     re$receivemsg
                                else
                                jmp     error$return
                                endif

                        receive:
        014F 2AAD00             lhld    msg$adr
        0152 1664               mvi     d,timeout$retries
                        receive$firstchar:
        0154 CD5E00             call    xcharin
        0157 D26201             jnc     got$firstchar
        015A 15                 dcr     d
        015B C25401             jnz     receive$firstchar
        015E E1                 pop     h               ; discard receive$retry rtn adr
        015F C34701             jmp     receive$timeout
                        got$firstchar:
        0162 E67F               ani     7fh
        0164 FE05               cpi     ENQ             ; Enquire?
        0166 C24F01             jnz     receive

        0169 0E06               mvi     c,ACK
        016B CD3F00             call    nChar$out       ; acknowledge ENQ with an ACK

        016E CD6300             call    Char$in
        0171 D8                 rc                      ; return to receive$retry
        0172 E67F               ani     7fh
        0174 FE01               cpi     SOH             ; Start of Header ?
        0176 C0                 rnz                     ; return to receive$retry
        0177 57                 mov     d,a             ; initialize the HCS
        0178 1E05               mvi     e,5
        017A CD8500             call    Msg$in
        017D D8                 rc                      ; return to receive$retry
        017E CD9000             call    Net$in
```

```
0181 D8                 rc                          ; return to receive$retry
0182 C2CD01             jnz     bad$checksum
0185 CDC501             call    send$ACK
0188 CD6300             call    Char$in
018B D8                 rc                          ; return to receive$retry
018C E67F               ani     7fh
018E FE02               cpi     STX                 ; Start of Data ?
0190 C0                 rnz                         ; return to receive$retry
0191 57                 mov     d,a                 ; initialize the CKS
0192 2B                 dcx     h
0193 5E                 mov     e,m
0194 23                 inx     h
0195 1C                 inr     e
0196 CD8500             call    msg$in              ; get DB0 DB1 ...
0199 D8                 rc                          ; return to receive$retry
019A CD6300             call    Char$in             ; get the ETX
019D D8                 rc                          ; return to receive$retry
019E E67F               ani     7fh
01A0 FE03               cpi     ETX
01A2 C0                 rnz                         ; return to receive$retry
01A3 82                 add     d
01A4 57                 mov     d,a                 ; update CKS with ETX
01A5 CD9000             call    Net$in              ; get CKS
01A8 D8                 rc                          ; return to receive$retry
01A9 CD6300             call    Char$in             ; get EOT
01AC D8                 rc                          ; return to receive$retry
01AD E67F               ani     7fh
01AF FE04               cpi     EOT
01B1 C0                 rnz                         ; return to receive$retry
01B2 7A                 mov     a,d
01B3 B7                 ora     a                   ; test CKS
01B4 C2CD01             jnz     bad$checksum
01B7 E1                 pop     h                   ; discard receive$retry rtn adr
01B8 2AAD00             lhld    msg$adr
01BB 23                 inx     h
01BC 3A0100             lda     configtbl+1
01BF 96                 sub     m
01C0 CAC501             jz      send$ACK            ; jump with A=0 if DID ok
01C3 3EFF               mvi     a,0ffh              ; return code shows bad DID
                send$ACK:
01C5 F5                 push    psw                 ; save return code
01C6 0E06               mvi     c,ACK
01C8 CD3F00             call    nChar$out           ; send ACK if checksum ok
01CB F1                 pop     psw                 ; restore return code
01CC C9                 ret

                bad$DID:
                bad$checksum:
01CD 0E15               mvi     c,NAK
01CF C34E00             jmp     Char$out            ; send NAK on bad chksm & not max retries
                ;       ret

                error$return:
01D2 210000             lxi     h,network$status
```

```
01D5 B6                 ora     m
01D6 77                 mov     m,a
01D7 CDDD01             call    ntwrkerror      ; perform any required device re-init.
01DA 3EFF               mvi     a,0ffh
01DC C9                 ret

                ntwrkerror:
                                                ;  perform any required device
01DD C9                 ret                     ;     re-initialization

                        page
```

```

                        ;
                        ntwrkwboot:

                        ;       This procedure is called each time the CCP is
                        ;       reloaded from disk.  This version prints ""
                        ;       on the console and then returns, but anything necessary
                        ;       for restart can be put here.

      01DE 0E09                 mvi     c,9
      01E0 111500               lxi     d,wboot$msg
      01E3 C30500               jmp     BDOS

                                page
```

```
                if      cpnos
        ;
        ;       LOGIN to a Master
        ;
        ; Equates
        ;
        buff    equ     0080h
        readbf  equ     10
        active  equ     0001$0000b
        loginpr:
                mvi     c,initpasswordmsglen
                lxi     h,initpasswordmsg
                lxi     d,passwordmsg
        copypassword:
                mov     a,m
                stax    d
                inx     h
                inx     d
                dcr     c
                jnz     copypassword
                mvi     c,print
                lxi     d,loginmsg
                call    BDOS
                mvi     c,readbf
                lxi     d,buff-1
                mvi     a,50h
                stax    d
                call    BDOS
                lxi     h,buff
                mov     a,m     ; get # chars in the command tail
                ora     a
                jz      dologin ; default login if empty command tail
                mov     c,a     ; A = # chars in command tail
                xra     a
                mov     b,a     ; B will accumulate master ID
        scanblnks:
                inx     h
                mov     a,m
                cpi     ' '
                jnz     pastblnks ; skip past leading blanks
                dcr     c
                jnz     scanblnks
                jmp     prelogin ; jump if command tail exhausted
        pastblnks:
                cpi     '['
                jz      scanMstrID
                mvi     a,8
                lxi     d,passwordmsg+5+8-1
                xchg
        spacefill:
```

```
        mvi     m,' '
        dcx     h
        dcr     a
        jnz     spacefill
        xchg
scanLftBrkt:
        mov     a,m
        cpi     '['
        jz      scanMstrID
        inx     d
        stax    d       ;update the password
        inx     h
        dcr     c
        jnz     scanLftBrkt
        jmp     prelogin
scanMstrID:
        inx     h
        dcr     c
        jz      loginerr
        mov     a,m
        cpi     ']'
        jz      prelogin
        sui     '0'
        cpi     10
        jc      updateID
        adi     ('0'-'A'+10) and 0ffh
        cpi     16
        jnc     loginerr
updateID:
        push    psw
        mov     a,b
        add     a
        add     a
        add     a
        add     a
        mov     b,a     ; accum * 16
        pop     psw
        add     b
        mov     b,a
        jmp     scanMstrID

prelogin:
        mov     a,b

dologin:
        lxi     b,passwordmsg+1
        stax    b
        dcx     b
        call    sendmsg
        inr     a
        lxi     d,loginfailedmsg
        jz      printmsg
        lxi     b,passwordmsg
        call    receivemsg
```

```
                inr     a
                lxi     d,loginfailedmsg
                jz      printmsg
                lda     passwordmsg+5
                inr     a
                jnz     loginOK
                jmp     printmsg

        loginerr:
                lxi     d,loginerrmsg
        printmsg:
                mvi     c,print
                call    BDOS
                jmp     loginpr         ; try login again

        loginOK:
                lxi     h,network$status ; HL = status byte addr
                mov     a,m
                ori     active  ; set active bit true
                mov     m,a
                ret

        ;
        ; Local Data Segment
        ;
        loginmsg:
                db      cr,lf
                db      'LOGIN='
                db      '$'

        initpasswordmsg:
                db      00h     ; FMT
                db      00h     ; DID Master ID #
                db      slave$ID ;SID
                db      40h     ; FNC
                db      7       ; SIZ
                db      'PASSWORD' ; password
        initpasswordmsglen equ  $-initpasswordmsg


        loginerrmsg:
                db      lf
                db      'Invalid LOGIN'
                db      '$'

        loginfailedmsg:
                db      lf
                db      'LOGIN Failed'
                db      '$'

                DSEG
        passwordmsg:
                ds      1       ; FMT
                ds      1       ; DID
```

```
                    ds      1       ; SID
                    ds      1       ; FNC
                    ds      1       ; SIZ
                    ds      8       ; DAT = password
                    endif

    01E6            end
```

```
ACK              0006   175#  559   618   672
ACTIVE           0010   164#  721#  836
ALTOS            FFFF    28#   62   219   266   296   338   347   464
ALWAYSRETRY      FFFF    30#  569   596
ASCII            0000    34#  247   306   370   400
BADCHECKSUM      01CD   632   662   678#
BADDID           01CD   677#
BDOS             0005    40    42#  707   736   741   830
CHARIN           0063   308   333#  414   522   555   621   634   646   655
CHARIN0          0065   331   344#  357
CHARIN1          0067   346#  355
CHARIN2          007C   353   360#
CHARINTIMEOUT    012B   526   566#
CHAROUT          004E   253   257   294#  384   519   680
CHKSIN           0094   418#
CNFGTBLADR       00C3    50   501#
CONFIGTBL        0000   114#  447   459   502   666
CONOUT           0002   180#
CPNOS            0000    24#   39    66   445   478   711
CR               000D   177#  844
DB82             0000    27#   56   187   274
DEBUG            0000    36#
DEFAULTMASTER    0000    97#
DELAY            002F   236#  302
DELAY1           0031   238#  244
DELAY2           0033   240#  242
DPRTI            001E   190#  203#  211#  222#  361
DPRTO            001E   196#  207#  215#  226#  291
DSC2             0000    26#   59   199   280   471
ENQ              0005   174#  518   615
ENQRESPONSE      00DB   521#  525
EOT              0004   173#  549   658
ERRORRETURN      01D2   570   573   597   600   683#
ETX              0003   172#  543   649
FALSE            0000    21#   22    24    26    27    32    34    36
FIRSTPASS        00B0   159#
GETACK           0118   536   551   554#
GETACK0          011E   528   557#
GOTENQRESPONSE   00E8   523   527#
GOTFIRSTCHAR     0162   608   613#
INITOK           00B6   482#
LF               000A   176#  844   859   864
LOGIN            0040   183#
MAXRETRIES       000A   155#  514   584
MODEM            0000    32#  150   200   335   471
MSGADR           00AD   148#  512   517   582   604   664
MSGIN            0085   387#  394   628   644
MSGOUT           009A   426#  531   542
MSGOUTLOOP       009F   432#  437
MSKI             0001   189#  202#  210#  221#  352
MSKO             0004   193#  206#  214#  225#  271
NAK              0015   178#  679
NCHAROUT         003F   265#  272   277   295   550   619   673
NETIN            0090   389   397#  630   653
```

```
CP/M RMAC ASSEM 1.1     #023    REQUESTER NETWORK I/O SYSTEM FOR CP/NET 1.2

NETOUT           007F   364#  435   535   548
NETWORKERRMSG    0021   103#
NETWORKSTATUS    0000   116#  317   319   491   494   684   834
NIOS             0000    45#   46
NTWRKERROR       01DD    53   687   691#
NTWRKINIT        00A9    48   443#
NTWRKSTS         00B8    49   490#
NTWRKWBOOT       01DE    54   698#
PRECHAROUT       003C   260#  431   544
PRINT            0009   181#  734   829
RCVERR           0002   165#  318   493   594
RCVMSG           0043   182#
RECALL           013D   586#  592
RECEIVE          014F   587   603#  616
RECEIVEFIRSTCHAR 0154   606#  610
RECEIVEMSG       0133    52   579#  817
RECEIVERETRY     0140   589#
RECEIVETIMEOUT   0147   593#  612
RERECEIVEMSG     0138   583#  598
RESENDMSG        00CC   513#  571
RETRYCOUNT       00AF   156#  515   563   585   590
SEND             00D1   516#  565
SENDACK          01C5   633   668   670#
SENDERR          0001   166#  493   567
SENDMSG          00C7    51   509#  812
SENDRETRY        0123   556   561#
SLAVEID          0056    57#   60#   63#   70    92   142   458   851
SOH              0001   170#  529   624
STATI            001F   188#  201#  209#  220#  349   351
STATO            001F   192#  205#  213#  224#  268   270   473   475
STX              0002   171#  541   637
TIMEOUTRETRIES   0064   151#  153#  520   605
TRUE             FFFF    22#   28    30
WBOOTMSG         0015    99#  706
XCHARIN          005E   329#  607
```

Listing E-1: Request Network I/O System

E.5 Modifying the NETWRKIF

The NETWRKIF, designed for an Altos ACS 8000-10, is also easy to modify. The NETWRKIF implements the protocol by checking for the first character of an incoming message through one of the XIOS CONIN routines. After receiving the first character and validating it, the NETWRKIF disables interrupts and reads the rest of the message in under direct process control. If an XIOS CONIN routine does not exist for the port to be used for the network, you must write one.

To modify this NETWRKIF, follow these steps:

1. Set the NMB$SLVS equate to the number of requesters to be supported. If more than four must be supported, you must add more Process Descriptors and queues.
2. If the server can only transmit or receive one message at a time, then the NETWRKIF supports a mutual exclusion queue to prevent collisions. To use this queue, set MUTEXIN or MUTEXOUT to true.
3. If the server is running on a Z80 processor, set Z80 to true for more efficient implementation of character I/O.
4. If all or some of the network RS-232 ports support only 7 bit ASCII, modify the BINARYASCII table by setting the appropriate entries to 0.
5. Modify the network port definitions. CONSOLE4$STATUS through PRINTER2$STATUS must be modified. Also, CHARIOTBL must be modified, so that the console numbers associated with the ports listed in STATUS$PORTS match.
6. I/O port numbers in the routines CHAR$OUT and CHAR$IN might have to be modified. You might

have to implement a I/O port table similar to STATUS$PORTS. This implementation relies on the fact that the Altos ACS 8000-10 always positions its I/O ports at a fixed offset from its status ports.

The sample NETWRKIF contains a debug conditional assembly flag that permits generation of a NETWRKIF.COM file. The NETWRKIF.COM version can debug a single requester, as follows:

1. Perform a GENSYS in which the SERVER.RSP is included; do not include a NETWRKIF.RSP. During the GENSYS, do not specify bank-switched memory.
2. Execute the MPM.SYS produced from GENSYS, and load the NETWRKIF.COM file with DDT, SID, or ZSID.
3. Use DDT, SID, or ZSID to debug the NETWRKIF process. This works only for a single requester.

```
CP/M RMAC ASSEM 1.1     #001    MASTER NETWORK I/F MODULE

                                title   'Master Network I/F Module'
                                page    54

                        ;*************************************************************
                        ;*************************************************************
                        ;**                                                         **
                        ;**      S e r v e r   N e t w o r k   I / F   M o d u l e   **
                        ;**                                                         **
                        ;*************************************************************
                        ;*************************************************************

                        ;/*
                        ;  Copyright (C) 1980
                        ;  Digital Research
                        ;  P.O. Box 579
                        ;  Pacific Grove, CA 93950
                        ;
                        ;  Modified October 5, 1982
                        ;
                        ;*/


         0000 =         false   equ     0
         FFFF =         true    equ     not false

         FFFF =         z80     equ     true

         0000 =         debug   equ     false
         0000 =         modem   equ     false

         0000 =         WtchDg  equ     false           ; include watch dog timer

         0000 =         mutexin equ     false           ; provide mutual exclusion on input
         0000 =         mutexout equ    false           ; provide mutual exclusion on output


                                if      debug

                        NmbSlvs equ     1               ;debug only one requester

                                lxi     sp,NtwrkIS0+2eh
                                mvi     c,145
                                mvi     e,64
                                call    bdos            ; set priority to 64
                                lxi     h,UQCBNtwrkQI0  ; initialize reentrant variables
                                lxi     d,UQCBNtwrkQO0
                                lxi     b,BufferQ0
                                mvi     a,00h
                                ret

                        bdosadr:
                                dw      0005h

                                else

CP/M RMAC ASSEM 1.1     #002    MASTER NETWORK I/F MODULE
```

```
0002 =         NmbSlvs equ     2               ;RSP is configured for two requesters
               bdosadr:
0000 0000              dw      $-$             ;XDOS entry point for RSP version
                       endif
               ;  Network Interface Process #0
               NtwrkIP0:
0002 0000              dw      0               ; link
0004 00                db      0               ; status
0005 40                db      64              ; priority
0006 6400              dw      NtwrkIS0+46     ; stack pointer
0008 4E7477726B        db      'NtwrkIP0'      ; name
0010 00                db      0               ; console
0011 FF                db      0ffh            ; memseg
0012                   ds      2               ; b
0014                   ds      2               ; thread
0016                   ds      2               ; buff
0018                   ds      1               ; user code & disk slct
0019                   ds      2               ; dcnt
001B                   ds      1               ; searchl
001C                   ds      2               ; searcha
001E                   ds      2               ; active drives
0020 0000              dw      0               ; HL'
0022 0000              dw      0               ; DE'
0024 0000              dw      0               ; BC'
0026 0000              dw      0               ; AF'
0028 0000              dw      0               ; IY
002A 0000              dw      0               ; IX
002C 8000              dw      UQCBNtwrkQI0    ; HL
002E A000              dw      UQCBNtwrkQO0    ; DE
0030 A600              dw      BufferQ0        ; BC
0032 0000              dw      0               ; AF, A = ntwkif console dev #
0034                   ds      2               ; scratch
               NtwrkIS0:
0036 C7C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
003E C7C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0046 C7C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
004E C7C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0056 C7C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
005E C7C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h
0064 4206              dw      setup
               QCBNtwrkQI0:
0066                   ds      2               ; link
0068 4E7477726B        db      'NtwrkQI0'      ; name
0070 0200              dw      2               ; msglen
0072 0100              dw      1               ; nmbmsgs
0074                   ds      2               ; dqph
0076                   ds      2               ; nqph
```



```
0078                   ds      2               ; msgin
007A                   ds      2               ; msgout
007C                   ds      2               ; msgcnt
007E                   ds      2               ; buffer
               UQCBNtwrkQI0:
0080 6600              dw      QCBNtwrkQI0     ; pointer
0082 8400              dw      BufferQI0Addr   ; msgadr
               BufferQI0Addr:
0084 A600              dw      BufferQ0
               QCBNtwrkQO0:
0086                   ds      2               ; link
0088 4E7477726B        db      'NtwrkQO0'      ; name
0090 0200              dw      2               ; msglen
0092 0100              dw      1               ; nmbmsgs
0094                   ds      2               ; dqph
0096                   ds      2               ; nqph
0098                   ds      2               ; msgin
```

```
009A              ds      2               ; msgout
009C              ds      2               ; msgcnt
009E              ds      2               ; buffer

          UQCBNtwrkQO0:
00A0 8600         dw      QCBNtwrkQO0     ; pointer
00A2 A400         dw      BufferQO0Addr   ; msgadr
          BufferQO0Addr:
00A4              ds      2

          BufferQ0:
00A6              ds      1               ; FMT
00A7              ds      1               ; DID
00A8              ds      1               ; SID
00A9              ds      1               ; FNC
00AA              ds      1               ; SIZ
00AB              ds      257             ; MSG

          ;       Network Interface Process #1

                  if      NmbSlvs GE 2
          NtwrkIP1:

                  if      NmbSlvs GE 3
                  dw      NtwrkIP2        ; link
                  else
01AC 0000         dw      0               ; link
                  endif

01AE 00           db      0               ; status
01AF 40           db      64              ; priority
01B0 0E02         dw      NtwrkIS1+46     ; stack pointer
01B2 4E7477726B   db      'NtwrkIP1'      ; name
01BA 00           db      0               ; console
01BB FF           db      0ffh            ; memseg
```



```
01BC              ds      2               ; b
01BE              ds      2               ; thread
01C0              ds      2               ; buff
01C2              ds      1               ; user code & disk slct
01C3              ds      2               ; dcnt
01C5              ds      1               ; searchl
01C6              ds      2               ; searcha
01C8              ds      2               ; active drives
01CA 0000         dw      0               ; HL'
01CC 0000         dw      0               ; DE'
01CE 0000         dw      0               ; BC'
01D0 0000         dw      0               ; AF'
01D2 0000         dw      0               ; IY
01D4 0000         dw      0               ; IX
01D6 2A02         dw      UQCBNtwrkQI1    ; HL
01D8 4A02         dw      UQCBNtwrkQO1    ; DE
01DA 5002         dw      BufferQ1        ; BC
01DC 0001         dw      0100h           ; AF, A = ntwkif console dev #
01DE              ds      2               ; scratch

          NtwrkIS1:
01E0 C7C7C7C7C7   dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
01E8 C7C7C7C7C7   dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
01F0 C7C7C7C7C7   dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
01F8 C7C7C7C7C7   dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0200 C7C7C7C7C7   dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0208 C7C7C7C7C7   dw      0c7c7h,0c7c7h,0c7c7h
020E 6906         dw      init

          QCBNtwrkQI1:
0210              ds      2               ; link
0212 4E7477726B   db      'NtwrkQI1'      ; name
021A 0200         dw      2               ; msglen
021C 0100         dw      1               ; nmbmsgs
021E              ds      2               ; dqph
0220              ds      2               ; nqph
0222              ds      2               ; msgin
0224              ds      2               ; msgout
0226              ds      2               ; msgcnt
```

```
0228                ds      2               ; buffer

               UQCBNtwrkQI1:
022A 1002           dw      QCBNtwrkQI1     ; pointer
022C 2E02           dw      BufferQI1Addr   ; msgadr
               BufferQI1Addr:
022E 5002           dw      BufferQ1

               QCBNtwrkQO1:
0230                ds      2               ; link
0232 4E7477726B     db      'NtwrkQO1'      ; name
023A 0200           dw      2               ; msglen
023C 0100           dw      1               ; nmbmsgs
023E                ds      2               ; dqph
0240                ds      2               ; nqph
```



```
0242                ds      2               ; msgin
0244                ds      2               ; msgout
0246                ds      2               ; msgcnt
0248                ds      2               ; buffer

               UQCBNtwrkQO1:
024A 3002           dw      QCBNtwrkQO1     ; pointer
024C 4E02           dw      BufferQO1Addr   ; msgadr
               BufferQO1Addr:
024E                ds      2

               BufferQ1:
0250                ds      1               ; FMT
0251                ds      1               ; DID
0252                ds      1               ; SID
0253                ds      1               ; FNC
0254                ds      1               ; SIZ
0255                ds      257             ; MSG
                    endif

               ;       Network Interface Process #2

                    if      NmbSlvs GE 3
               NtwrkIP2:

                    if      NmbSlvs GE 4
                    dw      NtwrkIP3        ; link
                    else
                    dw      0               ; link
                    endif

                    db      0               ; status
                    db      64              ; priority
                    dw      NtwrkIS2+46     ; stack pointer
                    db      'NtwrkIP2'      ; name
                    db      0               ; console
                    db      0ffh            ; memseg
                    ds      2               ; b
                    ds      2               ; thread
                    ds      2               ; buff
                    ds      1               ; user code & disk slct
                    ds      2               ; dcnt
                    ds      1               ; searchl
                    ds      2               ; searcha
                    ds      2               ; active drives
                    dw      0               ; HL'
                    dw      0               ; DE'
                    dw      0               ; BC'
                    dw      0               ; AF'
                    dw      0               ; IY
                    dw      0               ; IX
                    dw      UQCBNtwrkQI2    ; HL
                    dw      UQCBNtwrkQO2    ; DE
                    dw      BufferQ2        ; BC
```



```
                    dw      0200h           ; AF, A = ntwkif console dev #
```

```
                ds      2               ; scratch

NtwrkIS2:
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h
        dw      init

QCBNtwrkQI2:
        ds      2               ; link
        db      'NtwrkQI2'      ; name
        dw      2               ; msglen
        dw      1               ; nmbmsgs
        ds      2               ; dqph
        ds      2               ; nqph
        ds      2               ; msgin
        ds      2               ; msgout
        ds      2               ; msgcnt
        ds      2               ; buffer

UQCBNtwrkQI2:
        dw      QCBNtwrkQI2     ; pointer
        dw      BufferQI2Addr   ; msgadr
BufferQI2Addr:
        dw      BufferQ2

QCBNtwrkQO2:
        ds      2               ; link
        db      'NtwrkQO2'      ; name
        dw      2               ; msglen
        dw      1               ; nmbmsgs
        ds      2               ; dqph
        ds      2               ; nqph
        ds      2               ; msgin
        ds      2               ; msgout
        ds      2               ; msgcnt
        ds      2               ; buffer

UQCBNtwrkQO2:
        dw      QCBNtwrkQO2     ; pointer
        dw      BufferQO2Addr   ; msgadr
BufferQO2Addr:
        ds      2

BufferQ2:
        ds      1               ; FMT
        ds      1               ; DID
        ds      1               ; SID
        ds      1               ; FNC
        ds      1               ; SIZ
```



```
        ds      257             ; MSG
        endif

;       Network Interface Process #3

        if      NmbSlvs GE 4
NtwrkIP3:
        dw      0               ; link
        db      0               ; status
        db      64              ; priority
        dw      NtwrkIS3+46     ; stack pointer
        db      'NtwrkIP3'      ; name
        db      0               ; console
        db      0ffh            ; memseg
        ds      2               ; b
        ds      2               ; thread
        ds      2               ; buff
        ds      1               ; user code & disk slct
        ds      2               ; dcnt
        ds      1               ; searchl
        ds      2               ; searcha
```

```
                ds      2               ; active drives
                dw      0               ; HL'
                dw      0               ; DE'
                dw      0               ; BC'
                dw      0               ; AF'
                dw      0               ; IY
                dw      0               ; IX
                dw      UQCBNtwrkQI3    ; HL
                dw      UQCBNtwrkQO3    ; DE
                dw      BufferQ3        ; BC
                dw      0300h           ; AF, A = ntwkif console dev #
                ds      2               ; scratch

        NtwrkIS3:
                dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
                dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
                dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
                dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
                dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
                dw      0c7c7h,0c7c7h,0c7c7h
                dw      init

        QCBNtwrkQI3:
                ds      2               ; link
                db      'NtwrkQI3'      ; name
                dw      2               ; msglen
                dw      1               ; nmbmsgs
                ds      2               ; dqph
                ds      2               ; nqph
                ds      2               ; msgin
                ds      2               ; msgout
                ds      2               ; msgcnt
                ds      2               ; buffer
```



```

        UQCBNtwrkQI3:
                dw      QCBNtwrkQI3     ; pointer
                dw      BufferQI3Addr   ; msgadr
        BufferQI3Addr:
                dw      BufferQ3

        QCBNtwrkQO3:
                ds      2               ; link
                db      'NtwrkQO3'      ; name
                dw      2               ; msglen
                dw      1               ; nmbmsgs
                ds      2               ; dqph
                ds      2               ; nqph
                ds      2               ; msgin
                ds      2               ; msgout
                ds      2               ; msgcnt
                ds      2               ; buffer

        UQCBNtwrkQO3:
                dw      QCBNtwrkQO3     ; pointer
                dw      BufferQO3Addr   ; msgadr
        BufferQO3Addr:
                ds      2

        BufferQ3:
                ds      1               ; FMT
                ds      1               ; DID
                ds      1               ; SID
                ds      1               ; FNC
                ds      1               ; SIZ
                ds      257             ; MSG
                endif


                if      WtchDg
        ;  Watchdog Timer Process
        ;
        WatchDogPD:

                if      NmbSlvs GT 1
```

```
        dw      NtwrkIP1        ; link to the remaining NETWRKIF PD's
        else
        dw      0               ; link
        endif

        db      0               ; status
        db      64              ; priority
        dw      WatchDogSTK+46  ; stack pointer
        db      'WatchDog'      ; name
        db      0               ; console
        db      0ffh            ; memseg
        ds      2               ; b
        ds      2               ; thread
```

CP/M RMAC ASSEM 1.1   #009   MASTER NETWORK I/F MODULE

```
        ds      2               ; buff
        ds      1               ; user code & disk slct
        ds      2               ; dcnt
        ds      1               ; searchl
        ds      2               ; searcha
        ds      2               ; active drives
        dw      0               ; HL'
        dw      0               ; DE'
        dw      0               ; BC'
        dw      0               ; AF'
        dw      0               ; IY
        dw      0               ; IX
        dw      0               ; HL
        dw      0               ; DE
        dw      0               ; BC
        dw      0               ; AF
        ds      2               ; scratch

WatchDogSTK:
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
        dw      0c7c7h,0c7c7h,0c7c7h
        dw      WatchDog

WatchDogTime:
        dw      $-$     ; one-second counter

WatchDogTable:
;            Waiting Timeout   Start    Flag  Requester
        db      0,      0,      0,0,    0ah     ; #0
        db      0,      0,      0,0,    0bh     ; #1
        db      0,      0,      0,0,    0fh     ; #2
        db      0,      0,      0,0,    0dh     ; #3
        endif

        if      mutexin or mutexout
QCBMXSXmitq:                    ; MX queue for requester transmitting

        ds      2               ; link
        db      'MXSXmitq'      ; name
        dw      0               ; msglen
        dw      1               ; nmbmsgs
        ds      2               ; dqph
        ds      2               ; nqph
        ds      2               ; msgin
        ds      2               ; msgout
        ds      2               ; msgcnt
        ds      2               ; buffer (owner PD)

UQCBMXSXmitq:
        dw      QCBMXSXmitq
```

CP/M RMAC ASSEM 1.1   #010   MASTER NETWORK I/F MODULE

```
;       dw      0               ; no message, since it's an MX queue
;       db      'MXSXmitq'      ; no name, since the QCB pointer is resolved
        endif
```

```
                 ;       Server Configuration Table
                 configtbl:
0356 00                  db      0               ; Server status byte
0357 00                  db      0               ; Server ID
0358 02                  db      NmbSlvs         ; Maximum number of requesters supported
0359 00                  db      0               ; Number of requesters currently logged-in
035A 0000                dw      0000h           ; 16 bit vector of logged in requesters
035C                     ds      16              ; Requester ID's currently logged-in
036C 5041535357          db      'PASSWORD'      ; login password
0001 =           nmsg            equ     1       ; number of messages buffered
0096 =           slave$stk$len   equ     96h     ; server process stack size
                         if      NmbSlvs GE 2
                 slave1$stk:
0374                     ds      slave$stk$len-2
0408 0A04                dw      Slave1
                         endif
                         if      NmbSlvs GE 3
                 slave2$stk:
                         ds      slave$stk$len-2
                         dw      Slave2
                         endif
                         if      NmbSlvs GE 4
                 slave3$stk:
                         ds      slave$stk$len-2
                         dw      Slave3
                         endif
                         if      NmbSlvs GE 2
                 Slave1:
040A                     ds      52              ; SERVR1PR processor descriptor
                         endif
                         if      NmbSlvs GE 3
                 Slave2:
                         ds      52              ; SERVR2PR processor descriptor
                         endif
                         if      NmbSlvs GE 4
                 Slave3:
                         ds      52              ; SERVR3PR processor descriptor
                         endif
                 ;       Local Data Segment
```



```
                 BinaryASCII:
043E FF                  db      0ffh            ; Requester #0: 0=7 bit ASCII, FF=8 bit binary
043F FF                  db      0ffh            ;       #1
0440 FF                  db      0ffh            ;       #2
0441 FF                  db      0ffh            ;       #3
                 Networkstatus:
0442 00                  db      0               ; Slave #0 network status byte
0443 00                  db      0               ;       #1
0444 00                  db      0               ;       #2
0445 00                  db      0               ;       #3
0446 0000        conin:  dw      $-$             ; save area for XIOS routine address
000A =           max$retries     equ     10      ; maximum send message retries
                 ;
                 ;       The following tables are for use in the ALTOS i/o routines.
                 ;       Note that this program MUST be used with an XIOS which allows
                 ;       using the second printer port as a console port - Accessed as console
                 ;       #4
002B =           Console4$status equ     02bh
002F =           Console3$status equ     02fh
```

```
002D =         Console2$status equ     02dh
0029 =         Printer2$status equ     029h    ; ALSO CONSOLE #4

                       if      z80
               ;
               ;       ENTRIES IN THE FOLLOWING TWO TABLES MUST MATCH !!!!

               status$ports:
0448 2B                db      Console4$status ; Console 4 (Requester 0) status port
0449 2F                db      Console3$status ; Console 3 (Requester 1) status port
044A 2D                db      Console2$status ; Console 2 (Requester 2) status port
044B 29                db      Printer2$status ; Printer 2 (Requester 3) status port
                       endif

               chariotbl:                      ; Relationship between requesters and consoles
044C 03                db      3
044D 02                db      2
044E 01                db      1
044F 04                db      4

               ;       Network Status Byte Equates

0080 =         ntwrktxrdy      equ     10000000b       ; NETWRKIF ready to send msg
0010 =         active          equ     00010000b       ; requester logged into network
0008 =         msgerr          equ     00001000b       ; error in received message
0004 =         ntwrk           equ     00000100b       ; network alive
0002 =         msgovr          equ     00000010b       ; message overrun
0001 =         ntwrkrxrdy      equ     00000001b       ; NETWRKIF has rcvd msg

               ;       BDOS and XDOS Equates

```



```
0085 =         flagset equ     133             ; flag set
0086 =         makeq   equ     134             ; make queue
0089 =         readq   equ     137             ; read queue
008B =         writeq  equ     139             ; write queue
008D =         delay   equ     141             ; delay
008E =         dsptch  equ     142             ; dispatch
0090 =         createp equ     144             ; create process
009A =         sydatad equ     154             ; system data page address
0083 =         poll    equ     083h            ; Poll device

               ;       General Equates

0001 =         SOH     equ     01h             ; Start of Header
0002 =         STX     equ     02h             ; Start of Data
0003 =         ETX     equ     03h             ; End of Data
0004 =         EOT     equ     04h             ; End of Transmission
0005 =         ENQ     equ     05h             ; Enquire
0006 =         ACK     equ     06h             ; Acknowledge
000A =         LF      equ     0ah             ; Line Feed
000D =         CR      equ     0dh             ; Carriage Return
0015 =         NAK     equ     15h             ; Negative Acknowledge

0010 =         printer2        equ     10h     ; special poll device number for second
                                               ; printer port

               ;       Utility Procedures

               bdos:
0450 2A0000            lhld    bdosadr         ; get XDOS entry point from RSP start
0453 E9                pchl

               Nibout:                         ; A = nibble to be transmitted in ASCII
0454 FE0A              cpi     10
0456 D25F04            jnc     nibatof         ; jump if A-F
0459 C630              adi     '0'
045B 4F                mov     c,a
045C C36804            jmp     Charout
               nibatof:
045F C637              adi     'A'-10
0461 4F                mov     c,a
0462 C36804            jmp     Charout

               PreCharout:
```

```
0465 7A                mov     a,d
0466 81                add     c
0467 57                mov     d,a             ; update the checksum

                       if      z80             ; Z80 version, using OUT A,(C) instruction
               char$out:

               ;       Character output routine for network i/o
               ;       using the ALTOS SIO ports
               ;
               ;       Z80 version: this can use indirect port numbers in a clean,
```



```
               ;       reentrant fashion
               ;
               ;       Entry: C register contains 8 bit value to transmit
               ;       Entry : Slave number in register b

0468 E5                push    h
0469 D5                push    d
046A C5                push    b
046B 51                mov     d, c            ; save the character
046C 214804            lxi     h, status$ports
046F 48                mov     c, b
0470 0600              mvi     b, 0            ; set (BC) = (b)
0472 09                dad     b
0473 4E                mov     c,m

               ;       Now C contains the address of the correct status port

               outputloop:
0474 3E10              mvi     a,10h

               ;       out     (c),a
0476 ED79              db      0edh,79h

               ;       in      a,(c)
0478 ED78              db      0edh,78h

047A E604              ani     04h             ; wait for TXready
047C CA7404            jz      outputloop

               ;       In the Altos system, data registers are one below status registers...

047F 0D                dcr     c

               ;       out     (c),d
0480 ED51              db      0edh,51h

0482 C1                pop     b
0483 D1                pop     d
0484 E1                pop     h
0485 C9                ret

                       else

               char$out:

               ;       Character output routine for network I/O
               ;       using ALTOS SIO ports
               ;
               ;       8080 version: This has to dispatch and then use direct port I/O
               ;       --extremely messy to do reentrantly
               ;
               ;       Entry:  C = character to transmit
               ;               B = slave id byte
```



```
                       push    h
                       push    d
                       push    b

                       lxi     d,out0          ; dispatch address =
```

```
                        mov     l,b             ;   out0 + slaveid*16
                        mvi     h,0
                        dad     h
                        dad     h
                        dad     h
                        dad     h
                        dad     d
                        mvi     a,10h           ;load "get transmit status" value
                        pchl                    ;dispatch

                out0:
                        out     Console4$status ;wait for TXready status
                        in      Console4$status
                        ani     4
                        jz      out0

                        mov     a,c
                        out     Console4$status-1       ;write the character
                        pop     b
                        pop     d
                        pop     h
                        ret

                out1:   out     Console3$status
                        in      Console3$status
                        ani     4
                        jz      out1

                        mov     a,c
                        out     Console3$status-1
                        pop     b
                        pop     d
                        pop     h
                        ret

                out2:   out     Console2$status
                        in      Console2$status
                        ani     4
                        jz      out2

                        mov     a,c
                        out     Console2$status-1
                        pop     b
                        pop     d
                        pop     h
                        ret

                out3:   out     Printer2$status
                        in      Printer2$status
```



```
                        ani     4
                        jz      out3

                        mov     a,c
                        out     Printer2$status-1
                        pop     b
                        pop     d
                        pop     h
                        ret

                        endif


                Nibin:                          ; return nibble in A register
  0486 CDBD04           call    Charin
  0489 D8               rc
  048A E67F             ani     07fh
  048C D630             sui     '0'
  048E FE0A             cpi     10
  0490 DAA604           jc      Nibin$return    ; must be 0-9
  0493 C6F9             adi     ('0'-'A'+10) and 0ffh
  0495 FE10             cpi     16
  0497 DAA604           jc      Nibin$return    ; must be 10-15
  049A 3A4204           lda     networkstatus
  049D F608             ori     msgerr
```

```
049F 324204          sta     networkstatus
04A2 3E00            mvi     a,0
04A4 37              stc
04A5 C9              ret

             Nibin$return:
04A6 B7              ora     a
04A7 C9              ret

             xChar$in:                       ; Get the first character using polled
                                             ; console I/O. Note that the rest of the
                                             ; message will be received using direct
                                             ; port I/O with interrupts disabled.
                                             ; OVERRUNS ARE NOT POSSIBLE USING THIS SCHEME

04A8 E5              push    h
04A9 C5              push    b
04AA 21BA04          lxi     h, Charin$return
04AD E5              push    h
04AE 48              mov     c,b
04AF 0600            mvi     b,0
04B1 214C04          lxi     h, chariotbl
04B4 09              dad     b
04B5 56              mov     d, m            ; Get the console number
04B6 2A4604          lhld    conin
04B9 E9              pchl                    ; vector off

             Charin$return:
04BA C1              pop     b
```



```
04BB E1              pop     h
04BC C9              ret


                     if      z80
             char$in:

             ;       Character input routine for network i/o
             ;       using the ALTOS SIO ports at 125k baud
             ;
             ;       Z80 Version uses indirect port addresses loaded into register C
             ;
             ;       Entry : Slave number in register b
             ;       Exit  : Character in register a
             ;
04BD E5              push    h
04BE C5              push    b
04BF 214804          lxi     h, status$ports
04C2 48              mov     c, b
04C3 0600            mvi     b, 0            ; set (BC) = (b)
04C5 09              dad     b
04C6 4E              mov     c,m

             ;       Now C contains the address of the correct status port

04C7 2E50            mvi     l, 80

             inputloop1:
04C9 2D              dcr     l
04CA CADA04          jz      retout

             ;       in      a,(c)
04CD ED78            db      0edh,78h

04CF E601            ani     01h             ; wait for RXready
04D1 CAC904          jz      inputloop1

             ;       In the Altos system, data registers are one below status registers...

04D4 0D              dcr     c

             ;       in      a,(c)
04D5 ED78            db      0edh,78h        ;get the character

04D7 C1              pop     b
```

```
04D8 E1              pop     h
04D9 C9              ret

             retout:
04DA 37              stc                     ;set carry => error flag
04DB C1              pop     b
04DC E1              pop     h
04DD C9              ret

```



```
                     else

             char$in:

             ;       Character input routine for network I/O
             ;       using ALTOS SIO ports
             ;
             ;       8080 Version uses same nasty dispatch mechanism that the output
             ;       routine used
             ;
             ;       Entry:  B = Slave ID
             ;       Exit:   A = character input

                     push    h
                     push    d
                     push    b
                     lxi     d,in0           ; HL = in0 + 17*slaveid
                     mov     l,b
                     mvi     h,0
                     xchg
                     dad     d
                     xchg
                     dad     h
                     dad     h
                     dad     h
                     dad     h
                     dad     d

                     mvi     c,80            ; load status retry count
                     pchl                    ; dispatch

             in0:
                     dcr     c
                     jz      retout          ; error return if retry timeout

                     in      Console4$status ; wait for RXready
                     ani     1
                     jz      in0

                     in      Console4$status-1       ; get the character
                     pop     b
                     pop     d
                     pop     h
                     ret

             in1:
                     dcr     c
                     jz      retout

                     in      Console3$status
                     ani     1
                     jz      in1

                     in      Console3$status-1
```



```
                     pop     b
                     pop     d
                     pop     h
                     ret

             in2:
                     dcr     c
```

```
                        jz      retout

                        in      Console2$status
                        ani     1
                        jz      in2

                        in      Console2$status-1
                        pop     b
                        pop     d
                        pop     h
                        ret
                in3:
                        dcr     c
                        jz      retout

                        in      Printer2$status
                        ani     1
                        jz      in3

                        in      Printer2$status-1
                        pop     b
                        pop     d
                        pop     h
                        ret

                retout:                         ; error return (carry=1)
                        stc
                        pop     b
                        pop     d
                        pop     h
                        ret

                        endif


                Netout:                         ; C = byte to be transmitted
04DE 7A                 mov     a,d
04DF 81                 add     c
04E0 57                 mov     d,a
04E1 3A3E04             lda     BinaryASCII
04E4 B7                 ora     a
04E5 C26804             jnz     Charout         ; transmit byte in Binary mode
04E8 79                 mov     a,c
04E9 F5                 push    psw
04EA 1F                 rar
04EB 1F                 rar
04EC 1F                 rar
```



```
04ED 1F                 rar
04EE E60F               ani     0FH             ; Shift HI nibble to LO nibble
04F0 CD5404             call    Nibout
04F3 F1                 pop     psw
04F4 E60F               ani     0FH
04F6 C35404             jmp     Nibout

                Netin:                          ; byte returned in A register
                                                ; D  = checksum accumulator
04F9 3A3E04             lda     BinaryASCII
04FC B7                 ora     a
04FD CA0705             jz      ASCIIin
0500 CDBD04             call    charin          ;receive byte in Binary mode
0503 D8                 rc
0504 C31705             jmp     chksin

                ASCIIin:
0507 CD8604             call    Nibin
050A D8                 rc
050B 87                 add     a
050C 87                 add     a
050D 87                 add     a
050E 87                 add     a
050F F5                 push    psw
0510 CD8604             call    Nibin
0513 D8                 rc
0514 E3                 xthl
```

```
0515 B4                 ora     h
0516 E1                 pop     h
                chksin:
0517 B7                 ora     a
0518 F5                 push    psw
0519 82                 add     d               ; add & update checksum accum.
051A 57                 mov     d,a
051B F1                 pop     psw
051C C9                 ret

                Msgin:                          ; HL = destination address
                                                ; E  = # bytes to input
051D CDF904             call    Netin
0520 D8                 rc
0521 77                 mov     m,a
0522 23                 inx     h
0523 1D                 dcr     e
0524 C21D05             jnz     Msgin
0527 C9                 ret

                Msgout:                         ; HL = source address
                                                ; E  = # bytes to output
                                                ; D  = checksum
                                                ; C  = preamble character
0528 1600               mvi     d,0
052A CD6504             call    PreCharout

```



```
                Msgoutloop:
052D 4E                 mov     c,m
052E 23                 inx     h
052F CDDE04             call    Netout
0532 1D                 dcr     e
0533 C22D05             jnz     Msgoutloop
0536 C9                 ret

                ;       Network Initialization

                nwinit:

                ;       device initialization, as required


0537 3E47               mvi     a,047h          ;sets up CTC for baud rate of 125k
0539 D331               out     031h

                        if      nmbslvs ge 3    ;initialize only the ports that are needed
                        out     030h
                        endif

                        if      nmbslvs ge 4
                        out     032h
                        endif

053B 3E01               mvi     a,1             ;count of one => max speed
053D D331               out     031h

                        if      nmbslvs ge 3
                        out     030h
                        endif

                        if      nmbslvs ge 4
                        out     032h
                        endif


                ;       Find address of XIOS console output routine

053F 2A0100             lhld    0001h           ; get warmstart entry in the XIOS jump table
0542 23                 inx     h
0543 5E                 mov     e, m
0544 23                 inx     h
0545 56                 mov     d, m
0546 210600             lxi     h, 0006h        ; Offset for conin routine
0549 19                 dad     d
```

```
054A 224604          shld    conin           ; save the address
054D AF              xra     a               ; return code is 0=success
054E C9              ret


                ;       Network Status

```



```
                nwstat:                         ; C = Slave #
054F 0600            mvi     b,0
0551 214204          lxi     h,networkstatus
0554 09              dad     b
0555 7E              mov     a,m
0556 47              mov     b,a
0557 E6F5            ani     not (msgerr+msgovr)
0559 77              mov     m,a
055A 78              mov     a,b
055B C9              ret


                ;       Return Configuration Table Address

                cfgadr:
055C 215603          lxi     h,configtbl
055F C9              ret


                ;       Send Message on Network

                sndmsg:                         ; DE = message addr
                                                ;  C = Slave #
0560 41              mov     b,c
0561 3E0A            mvi     a,max$retries   ; A = max$retries

                send:
0563 F5              push    psw

                     if      mutexout

                ;       Use mutual exclusion if it is possible for some unsolicited input
                ;       to stomp on your output (This is nice is you;re running some sort
                ;       of multi-drop protocol)

                     push    b
                     push    d
                     mvi     c,readq
                     lxi     d,UQCBMXSXmitq
                     call    bdos            ; obtain mutual exclusion token
                     pop     d
                     pop     b
                     endif

0564 EB              xchg
0565 E5              push    h
0566 F3              di                      ; disable interrupts to avoid underrun
0567 0E05            mvi     c,ENQ
0569 CD6804          call    Charout         ; send ENQ
056C CDA005          call    getACK          ; won't return on an error
056F 1E05            mvi     e,5
0571 0E01            mvi     c,SOH
0573 CD2805          call    Msgout          ; send SOH FMT DID SID FNC SIZ
0576 AF              xra     a
```



```
0577 92              sub     d
0578 4F              mov     c,a
0579 CDDE04          call    Netout          ; send HCS (header checksum)
057C CDA005          call    getACK          ; won't return on an error
057F 2B              dcx     h
0580 5E              mov     e,m
0581 23              inx     h
0582 1C              inr     e
0583 0E02            mvi     c,STX
```

```
0585 CD2805            call    Msgout          ; send STX DB0 DB1 ...
0588 0E03              mvi     c,ETX
058A CD6504            call    PreCharout      ; send ETX
058D AF                xra     a
058E 92                sub     d
058F 4F                mov     c,a
0590 CDDE04            call    Netout          ; send CKS
0593 0E04              mvi     c,EOT
0595 CD6504            call    PreCharout      ; send EOT
0598 CDA005            call    getACK          ; won't return on an error
059B D1                pop     d               ; discard message address
059C F1                pop     psw             ; discard retry counter

                       if      mutexout
                       call    release$MX
                       endif

059D FB                ei                      ; return from suspended animation
059E AF                xra     a
059F C9                ret                     ; A = 0, successful send message

               getACK:
05A0 CDBD04            call    Charin
05A3 DAAB05            jc      getACK$timeout  ; receive timeout-->start error recovery
05A6 E67F              ani     7fh
05A8 D606              sui     ACK
05AA C8                rz

               getACK$timeout:
05AB D1                pop     d               ; discard return address

                       if      mutexout
                       push    b
                       call    release$MX
                       pop     b
                       endif

05AC D1                pop     d               ; DE = message address
05AD F1                pop     psw             ; A = retry count
05AE 3D                dcr     a
05AF C26305            jnz     send            ; continue if retry count non-zero
05B2 3D                dcr     a               ; else-->we're dead-->A = 0ffh
05B3 C9                ret                     ; failed to send message

                       if      mutexin or mutexout
```



```

               release$MX:                     ; send back requester transmit MX message
                       mvi     c,writeq
                       lxi     d,UQCBMXSXmitq
                       jmp     bdos
                       endif

               ;       Receive Message from Network

               rcvmsg:                         ; DE = message addr
                                               ;  C = Slave #
05B4 41                mov     b,c

               receive:
05B5 EB                xchg
05B6 E5                push    h
05B7 CDBF05            call    get$ENQ

               ;       a return to this point indicates an error

               receive$retry:
05BA FB                ei                      ; re-enable other processes

                       if      mutexin
                       push    b
                       call    release$MX
                       pop     b
                       endif

```

```
05BB D1                 pop     d
05BC C3B505             jmp     receive

               get$ENQ:                         ; get first character of message using
                                                ; polled console I/O
05BF CDA804             call    xCharin
05C2 DABF05             jc      get$ENQ
05C5 E67F               ani     7fh
05C7 FE05               cpi     ENQ             ; Start of Message ?
05C9 C2BF05             jnz     get$ENQ

                        if      mutexin

               ;        Don't get too involved with receiving a message if some other
               ;        NETWRKIF process is going to stomp you by sending a message along
               ;        the same line

                        push    b
                        push    h
                        mvi     c,readq
                        lxi     d,UQCBMXSXmitq
                        call    bdos
                        pop     h
                        pop     b
                        endif
```



```
05CC 0E06               mvi     c,ACK
05CE F3                 di                      ; requester in gear now serve only him

05CF CD6804             call    charout         ; send ACK to requester, allowing transmit
05D2 CDBD04             call    Charin
05D5 D8                 rc
05D6 E67F               ani     7fh
05D8 FE01               cpi     SOH
05DA C0                 rnz
05DB 57                 mov     d,a             ; initialize the HCS
05DC 1E05               mvi     e,5
05DE CD1D05             call    Msgin
05E1 D4F904             cnc     Netin
05E4 D8                 rc
05E5 7A                 mov     a,d
05E6 B7                 ora     a
05E7 C21406             jnz     sendNAK         ; jmp & send NAK if HCS <> 0
05EA 0E06               mvi     c,ACK
05EC CD6804             call    Charout
05EF CDBD04             call    Charin
05F2 D8                 rc
05F3 E67F               ani     7fh
05F5 FE02               cpi     STX
05F7 C0                 rnz
05F8 57                 mov     d,a             ; initialize the CKS
05F9 2B                 dcx     h
05FA 5E                 mov     e,m
05FB 23                 inx     h
05FC 1C                 inr     e
05FD CD1D05             call    msgin
0600 D4BD04             cnc     Charin
0603 D8                 rc
0604 E67F               ani     7fh
0606 FE03               cpi     ETX
0608 C0                 rnz
0609 82                 add     d
060A 57                 mov     d,a
060B CDF904             call    Netin           ; get Checksum byte
060E D8                 rc
060F 7A                 mov     a,d
0610 B7                 ora     a               ; should be zero
0611 CA1906             jz      sendACK         ; jump if checksum OK

               sendNAK:                         ; else-->refuse the message
0614 0E15               mvi     c,NAK
0616 C36804             jmp     Charout         ; send NAK and return to receive$retry

               sendACK:                         ; come here if message was received properly
```

```
0619 CDBD04           call    Charin          ; get EOT
061C D8               rc
061D E67F             ani     7fh
061F FE04             cpi     EOT
0621 C0               rnz
```



```
0622 0E06             mvi     c,ACK
0624 CD6804           call    Charout         ; send ACK if checksum ok
0627 D1               pop     d               ; discard return address
0628 D1               pop     d               ; discard message address
0629 FB               ei                      ; Dispense with the Rip Van Winkle act

                      if      mutexin
                      call    release$MX
                      endif

062A AF               xra     a
062B C9               ret


              restore:

              ;       This routine allows N copies of NtwrkIPx to run reentrantly.
              ;       It takes the values that were pre-initialized in the process
              ;       descriptor and later saved on the stack and loads them into
              ;       the registers, leaving the stack image untouched.  All variables
              ;       intrinsic to the process therefore always reside on the
              ;       process-dependent stack

062C F3               di                      ; this is a real critical region
062D E1               pop     h
062E 224006           shld    rtnadr
0631 E1               pop     h
0632 D1               pop     d
0633 C1               pop     b
0634 F1               pop     psw
0635 F5               push    psw
0636 C5               push    b
0637 D5               push    d
0638 E5               push    h
0639 E5               push    h
063A 2A4006           lhld    rtnadr
063D E3               xthl
063E FB               ei
063F C9               ret

0640          rtnadr: ds      2

                      if      WtchDg

              ;       WatchDog Timer Process
              ;       This process needs adjunct processes to handle the timeout flags
              ;       that it sets.  They might possibly abort the offending NtwrkIPx
              ;       process, recreate it, and allow it to re-initialize its queues

              WatchDog:
                      mvi     c,Delay
                      lxi     d,60            ; delay for 1 second
                      call    bdos
                      lhld    WatchDogTime
```



```
                      inx     h
                      shld    WatchDogTime
                      lxi     h,WatchDogTable-5
                      mvi     c,NmbSlvs

              WatchDogLoop:
                      lxi     d,0005h
                      dad     d
                      mov     a,m
                      ora     a
                      jz      WatchDogDec
```

```
                        inx     h
                        ana     m
                        dcx     h
                        jnz     WatchDogDec     ; waiting & timeout set
                        push    h               ; save HL -> WDT.waiting
                        inx     h
                        inx     h
                        di
                        mov     e,m
                        inx     h
                        mov     d,m
                        ei
                        lhld    WatchDogTime
                        mov     a,l
                        sub     e
                        mov     l,a
                        mov     a,h
                        sbb     d
                        mov     h,a
                        mvi     a,10            ; # seconds since started Charin
                        sub     l
                        mvi     a,0
                        sbb     h
                        pop     h
                        jnc     WatchDogDec
                        push    h
                        inx     h
                        mvi     m,0ffh          ; WDT.timeout = 0ffh
                        inx     h
                        inx     h
                        inx     h
                        push    b
                        mov     e,m             ; E = Flag #
                        mvi     c,Flagset
                        call    bdos
                        pop     b
                        pop     h

                WatchDogDec:
                        dcr     c
                        jnz     WatchDogLoop

                        jmp     WatchDog
```



```
                        endif


                ;       Setup code for Network Interface Procedures

                Setup:
 0642 F5                push    psw             ;create stack image of all reentrant variables
 0643 C5                push    b
 0644 D5                push    d
 0645 E5                push    h
 0646 CD3705            call    nwinit

                        if      mutexin or mutexout
                        mvi     c,makeq         ; make the mutual exclusion queue
                        lxi     d,QCBMXSXmitq
                        call    bdos

                        mvi     c,writeq        ; leave a token in the queue
                        lxi     d,UQCBMXSXmitq
                        call    bdos
                        endif

                        if      WtchDg
                        lxi     d,WatchDogPD    ;since this process is linked to all other
                                                ;NtwrkIPx processes, creating it creates all
                                                ;of the others
                        mvi     c,createp
                        call    bdos

                        else

```

```
                        if      NmbSlvs GE 2
 0649 11AC01            lxi     d,NtwrkIP1      ;this will create all the other NtwrkIPx
                                                ;processes if there's no watchdog
 064C 0E90              mvi     c,createp
 064E CD5004            call    bdos
                        endif
                        endif

 0651 0E8E              mvi     c,dsptch        ;give everything a chance to create its queues
 0653 CD5004            call    bdos

 0656 0E9A              mvi     c,sydatad
 0658 CD5004            call    bdos
 065B 110900            lxi     d,9
 065E 19                dad     d
 065F 115603            lxi     d,configtbl
 0662 73                mov     m,e
 0663 23                inx     h
 0664 72                mov     m,d             ; sysdatpage(9&10) = co.configtbl
                                                ; filling in the config tbl address is the
                                                ; the server processes' cue to start

                        if      modem
```



```
                ;       Initialize the modem

                        mvi     c,CR
                        mvi     b,slvmodem
                        call    Charout
                        mvi     c,'Z'
                        call    Charout
                        mvi     c,CR
                        call    Charout

                WtSpace:
                        call    Charin
                        jc      SetupDone
                        ani     07fh
                        cpi     ' '
                        jnz     WtSpace
                        mvi     c,'A'
                        call    Charout

                SetupDone:
                        endif

 0665 E1                pop     h
 0666 D1                pop     d
 0667 C1                pop     b
 0668 F1                pop     psw

                ;  Network Interface Reentrant Procedure

                Init:
 0669 F5                push    psw     ; A = network i/f console dev #
 066A C5                push    B       ; BC= buffer address
 066B D5                push    D       ; DE= UQCB ntwrk queue out
 066C E5                push    H       ; HL= UQCB ntwrk queue in
 066D 5E                mov     e,m
 066E 23                inx     h
 066F 56                mov     d,m
 0670 0E86              mvi     c,makeq
 0672 CD5004            call    bdos    ; make the ntwrk queue in
 0675 CD2C06            call    restore
 0678 EB                xchg
 0679 5E                mov     e,m
 067A 23                inx     h
 067B 56                mov     d,m
 067C 0E86              mvi     c,makeq
 067E CD5004            call    bdos    ; make the ntwrk queue out

                Loop:
 0681 CD2C06            call    restore
 0684 50                mov     d,b
 0685 59                mov     e,c
```

```
  1510
  1511   0686 4F                mov     c,a
  1512   0687 CDB405            call    rcvmsg
```

```
CP/M RMAC ASSEM 1.1     #029    MASTER NETWORK I/F MODULE

  1513
  1514   068A CD2C06            call    restore
  1515   068D EB                xchg
  1516   068E 0E8B              mvi     c,writeq
  1517   0690 CD5004            call    bdos
  1518
  1519   0693 CD2C06            call    restore
  1520   0696 0E89              mvi     c,readq
  1521   0698 CD5004            call    bdos
  1522
  1523   069B CD2C06            call    restore
  1524   069E 50                mov     d,b
  1525   069F 59                mov     e,c
  1526
  1527   06A0 4F                mov     c,a
  1528   06A1 CD6005            call    sndmsg
  1529
  1530   06A4 C38106            jmp     Loop
  1531
  1532   06A7                   end
```

```
CP/M RMAC ASSEM 1.1     #030    MASTER NETWORK I/F MODULE

ACK              0006   612# 1169  1244  1261  1297
ACTIVE           0010   587#
ASCIIIN          0507   984   989#
BDOS             0450    44   622# 1120  1193  1239  1349  1396  1420  1424  1432
                       1440  1445  1448  1497  1504  1517  1521
BDOSADR          0000    51#   58#  623
BINARYASCII      043E   541#  965   982
BUFFERQ0         00A6    47    89   118   138#
BUFFERQ1         0250   179   208   228#
BUFFERQI0ADDR    0084   116   117#
BUFFERQI1ADDR    022E   206   207#
BUFFERQO0ADDR    00A4   134   135#
BUFFERQO1ADDR    024E   224   225#
CFGADR           055C  1095#
CHARIN           04BD   771   816#  867#  985  1166  1248  1263  1274  1292  1470
CHARINRETURN     04BA   799   809#
CHARIOTBL        044C   578#  803
CHAROUT          0468   631   635   643#  692#  967  1129  1247  1262  1289  1298
                       1463  1465  1467  1476
CHKSIN           0517   987  1002#
CONFIGTBL        0356   493# 1096  1451
CONIN            0446   553#  806  1074
CONSOLE2STATUS   002D   564#  574   743   744   749   928   932
CONSOLE3STATUS   002F   563#  573   731   732   737   914   918
CONSOLE4STATUS   002B   562#  572   719   720   725   900   904
CR               000D   614# 1461  1466
CREATEP          0090   601# 1431  1439
DEBUG            0000    28#   37
DELAY            008D   599# 1347
DSPTCH           008E   600# 1444
ENQ              0005   611# 1128  1226
EOT              0004   610# 1151  1295
ETX              0003   609# 1145  1277
FALSE            0000    23#   24    28    29    31    33    34
FLAGSET          0085   595# 1395
GETACK           05A0  1130  1138  1153  1165#
GETACKTIMEOUT    05AB  1167  1172#
GETENQ           05BF  1205  1221# 1224  1227
INIT             0669   190   281   366  1488#
INPUTLOOP1       04C9   838#  846
LF               000A   613#
LOOP             0681  1506# 1530
MAKEQ            0086   596# 1418  1496  1503
MAXRETRIES       000A   555# 1105
MODEM            0000    29# 1458
MSGERR           0008   588#  781  1087
MSGIN            051D  1010# 1017  1255  1273
```

```
MSGOUT           0528  1020# 1133  1144
MSGOUTLOOP       052D  1027# 1032
MSGOVR           0002   590# 1087
MUTEXIN          0000    33#  471  1188  1212  1229  1303  1417
MUTEXOUT         0000    34#  471  1110  1157  1175  1188  1417
NAK              0015   615# 1288
NETIN            04F9   980# 1012  1256  1281
```

```
CP/M RMAC ASSEM 1.1     #031    MASTER NETWORK I/F MODULE

NETOUT           04DE   961# 1030  1137  1150
NETWORKSTATUS    0442   547#  780   782  1083
NIBATOF          045F   628   632#
NIBIN            0486   770#  990   997
NIBINRETURN      04A6   776   779   787#
NIBOUT           0454   626#  975   978
NMBSLVS          0002    39#   56#  148   151   239   242   330   419   496   505
                        512   518   524   529   534  1045  1049  1056  1060  1354
                       1436
NMSG             0001   502#
NTWRK            0004   589#
NTWRKIP0         0002    65#
NTWRKIP1         01AC   149#  420  1437
NTWRKIS0         0036    41    69    93#
NTWRKIS1         01E0   159   183#
NTWRKRXRDY       0001   591#
NTWRKTXRDY       0080   586#
NWINIT           0537  1037# 1415
NWSTAT           054F  1081#
OUTPUTLOOP       0474   666#  676
POLL             0083   603#
PRECHAROUT       0465   637# 1025  1146  1152
PRINTER2         0010   617#
PRINTER2STATUS   0029   565#  575   755   756   761   941   945
QCBNTWRKQI0      0066   102#  115
QCBNTWRKQI1      0210   192#  205
QCBNTWRKQO0      0086   120#  133
QCBNTWRKQO1      0230   210#  223
RCVMSG           05B4  1198# 1512
READQ            0089   597# 1118  1237  1520
RECEIVE          05B5  1202# 1219
RECEIVERETRY     05BA  1209#
RESTORE          062C  1311# 1498  1507  1514  1519  1523
RETOUT           04DA   840   859#  898   912   926   939   951#
RTNADR           0640  1322  1332  1337#
SEND             0563  1107# 1184
SENDACK          0619  1285  1291#
SENDNAK          0614  1260  1287#
SETUP            0642   100  1410#
SLAVE1           040A   508   525#
SLAVE1STK        0374   506#
SLAVESTKLEN      0096   503#  507   514   520
SNDMSG           0560  1102# 1528
SOH              0001   607# 1132  1251
STATUSPORTS      0448   571#  658   828
STX              0002   608# 1143  1266
SYDATAD          009A   602# 1447
TRUE             FFFF    24#   26
UQCBNTWRKQI0     0080    45    87   114#
UQCBNTWRKQI1     022A   177   204#
UQCBNTWRKQO0     00A0    46    88   132#
UQCBNTWRKQO1     024A   178   222#
WRITEQ           008B   598# 1191  1422  1516
WTCHDG           0000    31#  414  1339  1427
```

```
CP/M RMAC ASSEM 1.1     #032    MASTER NETWORK I/F MODULE

XCHARIN          04A8   791# 1223
Z80              FFFF    26#  567   642   815
```

Listing E-2: Server Network I/F Module

Appendix F
A CP/NET System for use with ULCnet

F.1 Overview of ULCnet

ULCnet® (Universal Low Cost Network) is a local area network system designed specifically for microcomputers in the CP/M and MP/M II operating system environments. ULCnet was introduced by Orange Compuco, Inc. in June 1982 as a low cost method of sharing resources and data among microcomputers of varying manufacture and architecture. ULCnet, in combination with CP/NET, creates a cost effective method for the development of shared data base applications among single user microcomputers. ULCnet architecture readily supports CP/NET implementation.

The ULCnet connector adaptor box can be connected to any computer that has a spare RS-232 port. ULCnet employs a multidrop topology with carrier sense, multiple-access design. Contention between network nodes is arbitrated using a full-duplex collision detection mechanism.

ULCnet is available to OEMs on a private label basis and through licensing. Keybrook Business Systems, Inc., Hayward, California, a licensee of ULCnet, produces the FileServer™ system. This system uses CP/NET to drive ULCnet. For more information on ULCnet, contact

Orange Compuco, Inc.
17801-G South East Main Street
Irvine, California 92714
(714) 957-8075

Orange Compuco distributes ULCnet connector adaptor hardware with a variety of release software, including the example programs in this appendix. In addition, Orange Compuco provides documentation detailing the installation and operation of ULCnet and logical structure of the data-link layer software. This documentation includes

- details on the installation and configuration of ULCnet
- a detailed description of the linkage between the proprietary data-link software and the user-definable Network I/O Drivers (NIOD)
- a detailed description of the interface between higher-level software and data-link software
- a description of the data-link interface (DLIF) between the data-link software and higher-level layers

F.2 Customizing a ULCnet SNIOS for the Requester

The CP/NET requester listing, SNIOS for ULCnet, that appears at the end of this section, is contained in a file called ULCNIOS.ASM on the CP/NET release disk and is designed to run ULCnet in a polled environment on a Xerox® 820 computer, now called the Xerox R820-IIS. The listing uses the ULCnet short format. This means that virtual circuit numbers must be agreed upon before the requester and the server can communicate. This version assumes that the server ID is always 0, and that up to four requesters, ID 1 through 4, are on the network. The virtual circuit number and the requester ID are always the same.

This SNIOS combines the two sections of the ULCnet protocol that are user configurable, the data-link interface (DLIF) and the network I/O drivers (NIOD). The DLIF acts as a transport layer between the NDOS and the data-link routines. The NIOD contains the physical device drivers use to communicate with the ULCnet network adaptor box. The bulk of the data-link protocol is contained in a module called PBMAIN.REL. This module is proprietary to Orange Compuco, and is therefore distributed only in REL file format by Orange Compuco.

When the NDOS instructs the SNIOS to send a message, the SNIOS first converts the CP/NET message format into ULCnet short format. The SNIOS then calls the TRANSMIT routine in PBMAIN to send the message, followed by the GETTCODE routine to discover the status of the message. If the send was successful, the SNIOS returns to the NDOS. If it was not successful, the SNIOS continues to try to send the message. No timeout is included in this routine to halt transmission.

To receive a message, the SNIOS calls RECEIVE, followed by GETRCODE to check the status of the message. If the status shows success, the message is converted from ULCnet format back into CP/NET format and returns to the NDOS. If the status shows an error, the SNIOS attempts to receive the message again.

To modify the SNIOS for a requester other than a Xerox 820, follow these steps:

1. Decide whether to make the requester operate in a polled or interrupt-driven environment. If you want interrupts, set the INTERRUPTS assembly switch to TRUE, and link the module using IPBMAIN instead of PBMAIN.
2. If your ULCnet connector adaptor has been modified for self clocked operation, set the assembly switch SLFCLKD to TRUE. Application notes detailing how to modify the connector adaptor for self-clocked operation are available from Orange Compuco.
3. Determine your requester's transmission speed capabilities. Set the baud rate masks BAUDSL and BAUDSH to reflect these values. Enter values for the requester's baud rate generator into the table BAUDTBL.
4. Modify the port numbers for the baud rate generator and the UART to reflect those used by your requester.
5. Modify the NIOD to run on your requester. The NIOD is currently set up to drive a Z80 SIO chip. If your requester has an SIO, it needs little modification. The routine PGMUART, which sets up the network port for ULCnet operation, might have to be modified. In an interrupt driven system, interrupt vectors must be set up here.
6. Assemble and link the SNIOS by performing

   ```
   A>RMAC ULCNIOS
   A>LINK SNIOS=ULCN1OS,PBMAIN[OS]
   ```

   If the requester is interrupt-driven, perform

   ```
   A>LINK SNIOS=ULCNIOS,IPBKAIN[OS]
   ```

   to link the module. The module is then ready for installation on the CP/NET requester system disk.

```
CP/M RMAC ASSEM 1.1     #001    REQUESTER NETWORK I/O SYSTEM FOR ULCNET

                                title 'Requester Network I/O System for ULCnet'
                                page    54

                        ;****************************************************************************
                        ;****************************************************************************
                        ;**                                                                        **
                        ;**                          SNIOS FOR ULCNET                              **
                        ;**                                                                        **
                        ;****************************************************************************
                        ;****************************************************************************

                        ;       Developed jointly by:
                        ;
                        ;               Digital Research, Inc.
                        ;               P.O. Box 579
```

```
                ;               Pacific Grove, CA 93950
                ;       and
                ;               Keybrook Business Systems, Inc.
                ;               2035 National Avenue
                ;               Hayward, CA 94545

                ;       This SNIOS was written for a Xerox 820 attached to Orange
                ;       Compuco's ULCnet network adaptor.  This module transports
                ;       messages between the NDOS and the low-level data-link software
                ;       provided by Orange Compuco.  It also contains the physical drivers
                ;       usually contained in the NIOD module.  This version is not
                ;       interrupt-driven and must be linked with PBMAIN.REL.



0000 =          false   equ     0
FFFF =          true    equ     not false

0000 =          interrupts      equ     false           ; false=polled, true=interrupt-driven
FFFF =          netstats        equ     true            ; switch to gather network statistics
FFFF =          slfclkd         equ     true            ; supports self-clocked operation

                ; Linkage information

                        public  setbaud,xmit,recv,initu ; NIOD routines called by IPBMAIN
                        public  inituart,pgmuart
                        public  chkstat,netidle,initrecv
                        public  wait,restuart,csniod
                        public  dsblxmit
                        public  dllbau,netadr

                        if      interrupts
                        public  enblrecv,dsblrecv
                        endif

                        extrn   transmit,receive        ; IPBMAIN routines and objects
                        extrn   gettcode,getrcode
                        extrn   csdll,dllon,regshrt
                        extrn   terrcnt,parcntr,ovrcntr
```



```
                        extrn   frmcntr,inccntr

                        if      interrupts
                        extrn   rtmochk                 ; IPBMAIN interrupt routines
                        extrn   dlisr,reisr,niisr
                        endif


                ; Hardware definitions for the Z80-SIO channel A - For the Xerox 820.

0003 =          baudsl  equ     03h             ; Usable baud rates: 9600, 19.2K asynch.,
002A =          baudsh  equ     2ah             ; 76.8K, 153.6K, 307.2K self-clocked

                                                ; baud rate capability mask
2A03 =          bauds   equ     (baudsh*100h)+baudsl

0000 =          baudgen equ     0               ; External baud rate generator register
0006 =          siocmd  equ     6               ; Command/Mode register
0006 =          siostat equ     6               ; Status register
0004 =          sioxmit equ     4               ; Transmit register
0004 =          siorecv equ     4               ; Receive register

0002 =          xrdybit equ     2               ; Transmit buffer empty status bit
0004 =          xrdymsk equ     4               ; transmit buffer empty status mask
0000 =          rrdybit equ     0               ; Receive buffer full status bit
0001 =          rrdymsk equ     1               ; receive buffer full status mask
0003 =          carbit  equ     3               ; Net Idle detect bit position
0008 =          carmsk  equ     8               ; Net Idle detect mask
0030 =          errst   equ     030h            ; Error flag reset
0070 =          errbits equ     070h            ; Error bit position mask
0004 =          pbit    equ     4               ; Parity error bit position
0010 =          pmsk    equ     10h             ; parity error mask
0005 =          obit    equ     5               ; Overrun error bit position
0020 =          omsk    equ     20h             ; overrun error mask
0006 =          fbit    equ     6               ; Framing error bit position
```

```
0040 =          fmsk    equ     40h             ; framing error mask
0003 =          selfbit equ     3               ; Self clock bit position
0008 =          selfmsk equ     8               ; slef clock bit mask
00EA =          dtron   equ     0eah            ; Turn on DTR
006A =          dtroff  equ     06ah            ; Turn off DTR
00C1 =          enarcv  equ     0c1h            ; Enable receive-clock
00C0 =          disrcv  equ     0c0h            ; Disable receive clock
000F =          enaslf  equ     00fh            ; Enable Self-clock mode
004F =          disslf  equ     04fh            ; Disable Self-clock mode

                ; SIO Mode 2 interrupts vector table

FF08 =          siov4   equ     0ff08h          ; SIO port A xmit buffer empty
FF0A =          siov5   equ     0ff0ah          ; SIO port A external status change
FF0C =          siov6   equ     0ff0ch          ; SIO port A receive
FF0E =          siov7   equ     0ff0eh          ; SIO port A special receive condition

                ; Message Buffer Offsets
```



```
0000 =          fmt             equ     0               ; format
0001 =          did             equ     fmt+1           ; destination ID
0002 =          sid             equ     did+1           ; source ID
0003 =          fnc             equ     sid+1           ; server function number
0004 =          siz             equ     fnc+1           ; size of message (normalized to 0)
0005 =          msg             equ     siz+1           ; message
0106 =          buf$len         equ     msg+257         ; length of total message buffer

                ; ULCnet Packet Offsets

0000 =          ulc$fmt         equ     0               ; packet format
0001 =          ulc$v$circ      equ     ulc$fmt+1       ; virtual circuit number
0002 =          ulc$len$lo      equ     ulc$v$circ+1    ; low order of length
0003 =          ulc$len$hi      equ     ulc$len$lo+1    ; high order of length
0004 =          ulc$fnc         equ     ulc$len$hi+1    ; start of message: function code
0005 =          ulc$msg         equ     ulc$fnc+1       ; CP/NET message

                ; Network Status Byte Equates

0010 =          active          equ     0001$0000b      ; slave logged in on network
0002 =          rcverr          equ     0000$0010b      ; error in received message
0001 =          senderr         equ     0000$0001b      ; unable to send message

                        CSEG
0005 =          BDOS    equ     0005h

                NIOS:
                        public  NIOS

                ; Jump vector for SNIOS entry points

0000 C3E100             jmp     ntwrkinit       ; network initialization
0003 C3EE00             jmp     ntwrksts        ; network status
0006 C3F600             jmp     cnfgtbladr      ; return config table addr
0009 C30401             jmp     sendmsg         ; send message on network
000C C32001             jmp     receivemsg      ; receive message from network
000F C3FA00             jmp     ntwrkerror      ; network error
0012 C30301             jmp     ntwrkwboot      ; network warm boot

0001 =          rqstr$id        equ     1       ; requester ID: must be between 1 and 4
004B =          fmt$byte        equ     4bh     ; format byte: short format with data-link
                                                ; acknowledge, 153.6K baud self-clocked

                        DSEG

                ; Transport Layer Data

                network$error$msg:
```

```
0000 0D0A             db      0dh,0ah
```



```
0002 4E6574776F       db      'Network Error'
000F 0D0A             db      0dh,0ah
0011 24               db      '$'


              ; Requester Configuration Table

              configtbl:
              Network$status:

0012                  ds      1               ; network status byte
0013 01               db      rqstr$id        ; slave processor ID number
0014                  ds      2               ; A:  Disk device
0016                  ds      2               ; B:   "
0018                  ds      2               ; C:   "
001A                  ds      2               ; D:   "
001C                  ds      2               ; E:   "
001E                  ds      2               ; F:   "
0020                  ds      2               ; G:   "
0022                  ds      2               ; H:   "
0024                  ds      2               ; I:   "
0026                  ds      2               ; J:   "
0028                  ds      2               ; K:   "
002A                  ds      2               ; L:   "
002C                  ds      2               ; M:   "
002E                  ds      2               ; N:   "
0030                  ds      2               ; O:   "
0032                  ds      2               ; P:   "
0034                  ds      2               ; console device
0036                  ds      2               ; list device:

              ; List Buffer Data

0038                  ds      1               ;       buffer index

0039 00               db      0               ;       FMT
003A 00               db      0               ;       DID
003B 01               db      rqstr$id        ;       SID
003C 05               db      5               ;       FNC
003D                  ds      1               ;       SIZ
003E                  ds      1               ;       MSG(0)  List number
003F                  ds      128             ;       MSG(1) ... MSG(128)


              ; ULCnet Data Definitions

00BF          netadr: ds      3               ;ULCnet network address
00C2          dllbau: ds      2               ;baud rate mask

0016 =        timeval equ     22              ; WAIT routine time constant
                                              ; 12 for 2.5 megahertz Z80
                                              ; 22 for 4.0 megahertz Z80

00C4 FF       curbaud db      0ffh            ; Current baud rate
```



```


                                              ; table to convert baud number codes
                                              ;   into a bit mask

00C5 0102040810btbl:   db      1,2,4,8,16,32,64,128


              baudtbl:                        ; async baud rate table

00CD 0E               db      0eh             ; 9600 Baud
00CE 0F               db      0fh             ; 19200

              scbaudt:                        ; self-clock baud rate table

```

```
00CF 00                 db      0               ;  62500 Baud - Not implemented
00D0 0D                 db      0dh             ;  76800 Baud
00D1 00                 db      0               ; 125000 Baud - Not implemented
00D2 0E                 db      0eh             ; 153600 Baud
00D3 00                 db      0               ; 250000 Baud - Not implemented
00D4 0F                 db      0fh             ; 307200 Baud

                        if      interrupts
               sioiblk db      030h,14h,4fh,15h,06ah,13h,0c1h,11h,01h,10h,10h,30h
                        else
00D5 30144F156Asioiblk db      030h,14h,4fh,15h,06ah,13h,0c1h,11h,00h,10h,10h,30h
                        endif

000C =         sioilen equ     $-sioiblk


                        page
```



```

               ;       Network Initialization Routine

               ntwrkinit:

00E1 CD0000             call    csdll                   ; cold start the data link
00E4 CD0000             call    dllon                   ; initialize the SIO drivers
00E7 3E01               mvi     a,rqstr$id              ; register the id with the data link
00E9 CD0000             call    regshrt
00EC AF                 xra     a                       ; return with no error
00ED C9                 ret


               ;       Return network status byte

               ntwrksts:

00EE 3A1200             lda     network$status
00F1 47                 mov     b,a
00F2 E6FC               ani     not (rcverr or senderr)
00F4 78                 mov     a,b
00F5 C9                 ret


               ;       Return configuration table address

               cnfgtbladr:

00F6 211200             lxi     h,configtbl
00F9 C9                 ret

               ;       Network error routine


               ntwrkerror:

00FA 0E09               mvi     c,9
00FC 110000             lxi     d,network$error$msg
00FF CD0500             call    bdos

0102 C9                 ret

               ;       Network Warm Boot Routine

               ntwrkwboot:                              ; this entry is unused in this version

0103 C9                 ret


               ;       Send a Message on the Network
               ;       Input:
               ;               BC=pointer to message buffer
               ;       Output:
               ;               A = 0 if successful
```

```
                ;                   1 if failure

                sendmsg:

0104 C5                 push    b
0105 60                 mov     h,b
0106 69                 mov     l,c

0107 364B               mvi     m,fmt$byte              ;set ulc$net format byte

0109 23                 inx     h                       ;reformat source to virtual circuit
010A 23                 inx     h
010B 56                 mov     d,m
010C 2B                 dcx     h
010D 72                 mov     m,d


010E 23                 inx     h
010F 23                 inx     h
0110 46                 mov     b,m                     ;save function

0111 23                 inx     h
0112 5E                 mov     e,m                     ;get size
0113 70                 mov     m,b                     ;function=msg(0) in ULC format

0114 1600               mvi     d,0
0116 13                 inx     d
0117 13                 inx     d                       ;normalize CP/NET to ULC sizes

0118 2B                 dcx     h
0119 72                 mov     m,d
011A 2B                 dcx     h
011B 73                 mov     m,e

011C C1                 pop     b                       ;restore buffer pointer

011D C34A01             jmp     dl$send                 ;blast away


                ;       Receive a Message on the Network
                ;
                ;       This routine calls the data-link routine to receive the message,
                ;       then converts it into ULCnet format.
                ;
                ;       Input:
                ;               BC = pointer to buffer to receive the message
                ;       Output:
                ;               A  = 0 if successful
                ;                    1 if failure

                receivemsg:

0120 C5                 push    b                       ;save buffer pointer

```



```
0121 CD3701             call    dl$receive              ;slurp the message

0124 E1                 pop     h
0125 3601               mvi     m,1                     ;FMT = 0 (requester to server)

0127 23                 inx     h                       ;DID already = virtual circuit #

0128 23                 inx     h                       ;get length
0129 5E                 mov     e,m
012A 23                 inx     h
012B 56                 mov     d,m

012C 1B                 dcx     d
012D 1B                 dcx     d                       ;normalize ULC to CP/NET format

012E 23                 inx     h
012F 7E                 mov     a,m                     ;save FNC

0130 73                 mov     m,e                     ;format SIZ (<256)
```

```
0131 2B                 dcx     h
0132 77                 mov     m,a                     ;format FNC

0133 2B                 dcx     h
0134 AF                 xra     a                       ;set success
0135 77                 mov     m,a                     ;assume server always 0

0136 C9                 ret                             ;CP/NET message formatted form ULCnet



                ; Data Link Interface Routines


                ; DL$RECEIVE:  Network Receive Function.
                ;       Input:
                ;               BC = Buffer address


                dl$receive:

0137 50                 mov     d,b             ; Buffer address in DE for data link
0138 59                 mov     e,c

                rretry:

0139 AF                 xra     a               ; Packet mode
013A 010101             lxi     b,257           ; Buffer size
013D 210000             lxi     h,0             ; Infinite wait
0140 D5                 push    d               ; Save buffer address for retry

0141 CD7801             call    psrecv          ; Initiate Receive and wait for completion

0144 D1                 pop     d               ; Restore buffer address
```



```
0145 B7                 ora     a
0146 C8                 rz                      ; Return if no error

0147 C33901             jmp     rretry          ; Jump to try again if error


                ; DL$SEND: Network Transmit Function
                ;       Input:
                ;               BC = Buffer address

                dl$send:

014A 50                 mov     d,b             ; Buffer address in DE for data link
014B 59                 mov     e,c

                tretry:

014C AF                 xra     a               ; Packet mode, wait for Net Idle
014D D5                 push    d               ; Save buffer address for retry

014E CD5701             call    psxmit          ; Initiate Transmit, wait for completion

0151 D1                 pop     d               ; Restore buffer address
0152 B7                 ora     a
0153 C8                 rz                      ; Return if no error

0154 C34C01             jmp     tretry          ; Jump to retry if error

                ; PSXMIT:  Transmit the packet pointed at by DE.  If carry flag is set
                ;          then don't wait for the Net to become idle.
                ;
                ;          Returns the completion code in A
                ;               0       - Transmission ok and Data Link Ack Received
                ;                         (In the case of multicast, no Ack required)
                ;               2       - Transmission OK but no Data Link Ack received.
                ;
                ;               4       - Other error.

                psxmit:
```

```
0157 CD0000           call    transmit                ; This will transmit, set return code
               twait:
015A CD0000           call    gettcode                ; A := GETTCODE - Xmit return code
015D 5F               mov     e,a
015E 1600             mvi     d,0
0160 216901           lxi     h,trtbl                 ; dispatch on the return code
0163 19               dad     d
0164 5E               mov     e,m
0165 23               inx     h
0166 66               mov     h,m
0167 6B               mov     l,e
0168 E9               pchl
```



```
               trtbl:
0169 7701             dw      psxret                  ; Good transmission
016B 7701             dw      psxret                  ; No Data Link Ack
016D 7701             dw      psxret                  ; Too many collisions
016F 7701             dw      psxret                  ; Transmitter is disabled
0171 5A01             dw      twait                   ; Transmitter is idle
0173 5A01             dw      twait                   ; Transmitter is in progress
0175 5A01             dw      twait                   ; Transmitter is waiting for ack
               psxret:
0177 C9               ret
               ; PSRECV:  Receive a packet into buffer pointed at by DE.  Length of
               ;          packet must be less than length of buffer in BC. HL is the receive
               ;          timeout count.
               ;
               ;          Upon return clear the carry bit if a packet received and ACKed.
               ;          Set the carry flag if any error occured.
               psrecv:
0178 CD0000           call    receive                 ; Receive.  Return code will be set
               rwait:
017B CD0000           call    getrcode                ; A := GETRCODE
017E 5F               mov     e,a
017F 1600             mvi     d,0
0181 218A01           lxi     h,rrtbl                 ; dispatch on the return code
0184 19               dad     d
0185 5E               mov     e,m
0186 23               inx     h
0187 66               mov     h,m
0188 6B               mov     l,e
0189 E9               pchl
               rrtbl:
018A 9601             dw      rgood                   ; Good receive
018C 9801             dw      rbad                    ; Bad receive
018E 9801             dw      rbad                    ; Disabled
                      if      not interrupts
0190 9801             dw      rbad                    ; Still idle after timeout
                      else
                      dw      ridle                   ; Idle
                      endif
0192 7B01             dw      rwait                   ; Inprogress
0194 7B01             dw      rwait                   ; In progress and for us.
```

```
                        if      interrupts
                ridle:

                        call    rtmochk                 ; Check for timeout
                        jc      ridle1                  ; Jump if timeout
                        call    wait1                   ; Wait 1 ms
                        jmp     rwait                   ; Continue to wait if no timeout

                ridle1:

                        call    dsblrecv                ; Disable the receiver
                        stc
                        ret                             ; Return with error
                        endif

                rgood:

      0196 A7           ana     a
      0197 C9           ret

                rbad:

      0198 37           stc                             ; Indicate error
      0199 C9           ret
                        page
```



```


                ; NIOD routines



                ; SETBAUD:  Set the baud rate based on the baud rate code in A.  Do special
                ;           logic for self-clocked mode.
                ;
                ;               0 = 9600 baud
                ;               1 = 19200 baud
                ;               9 = 76800 baud self-clock
                ;               11= 153600 baud self-clock
                ;               13= 307200 baud self-clock
                ;
                ; If this station cannot handle the requested baud rate, then set
                ; the carry flag.

                setbaud:

      019A E60F         ani     0fh             ; mask all but the baud bits
      019C 21C400       lxi     h,curbaud       ; are we at the current baud rate?
      019F BE           cmp     m
      01A0 C8           rz                      ; yes-->all done

      01A1 47           mov     b,a             ; else-->get baud rate generator value
      01A2 E607         ani     7
      01A4 5F           mov     e,a
      01A5 1600         mvi     d,0

      01A7 21C500       lxi     h,btbl          ; point to vertical-to-horizontal decode
      01AA 19           dad     d               ;   table

                        if      slfclkd
      01AB 78           mov     a,b
      01AC E608         ani     selfmsk         ; is this a self-clocked value?
      01AE C2D601       jnz     selfclkd
                        endif

      01B1 3E03         mvi     a,baudsl        ; get legal baud rate mask
      01B3 A6           ana     m
      01B4 37           stc
      01B5 C8           rz                      ; return with error if its an illegal rate

                        if      slfclkd
      01B6 3E05         mvi     a,5             ; else-->switch off possible self-clock mode
      01B8 D306         out     siocmd
      01BA 3E6A         mvi     a,dtroff        ; disable DTR in SIO register 5
      01BC D306         out     siocmd
```

```
01BE 3E04            mvi     a,4             ; disable sync mode in register 4
01C0 D306            out     siocmd
01C2 3E4F            mvi     a,disslf
01C4 D306            out     siocmd
```



```
                     endif
01C6 21CD00          lxi     h,baudtbl       ; point to async baud rate table
             outbau:
01C9 19              dad     d               ; get async baud rate value
01CA 7E              mov     a,m
01CB D300            out     baudgen         ; load it into the baud rate generator
                                             ; NOTE: This is not a CTC
01CD 21C400          lxi     h,curbaud
01D0 70              mov     m,b             ; set current baud byte
01D1 CDA702          call    wait            ; allow the system to reach equilibrium
01D4 A7              ana     a               ; return success
01D5 C9              ret
                     if      slfclkd
             ; Throw SIO into self-clocked mode
             selfclkd:
01D6 3E2A            mvi     a,baudsh        ; Is this a legal rate?
01D8 A6              ana     m
01D9 37              stc
01DA C8              rz                      ; return an error if not
01DB 3E04            mvi     a,4             ; enable sync mode in register 4
01DD D306            out     siocmd
01DF 3E0F            mvi     a,enaslf
01E1 D306            out     siocmd
01E3 3E05            mvi     a,5             ; enable DTR in register 5
01E5 D306            out     siocmd
01E7 3EEA            mvi     a,dtron
01E9 D306            out     siocmd
01EB 21CF00          lxi     h,scbaudt       ; point to baud rate table for self-clock mode
01EE C3C901          jmp     outbau          ; program the baud rate generator
                     endif
             ; DSBLXMIT:  Disable the transmitter if in self clocked mode
             dsblxmit:
                     if      slfclkd
01F1 3AC400          lda     curbaud         ; are we in self-clocked mode?
01F4 E608            ani     selfmsk
01F6 C8              rz                      ; no-->don't bother
01F7 3E05            mvi     a,5             ; disable SIO from transmitting by disabling
```



```
01F9 D306            out     siocmd          ; DTR in register 5
01FB 3E6A            mvi     a,dtroff
01FD D306            out     siocmd
01FF 3E05            mvi     a,5             ; Enable receive by re-enabling DTR
0201 D306            out     siocmd
0203 3EEA            mvi     a,dtron
0205 D306            out     siocmd
                     endif
0207 C9              ret
```

```
                ; XMIT:  Transmit the byte in A on network A.


                xmit:

                        if      not interrupts
0208 F5                 push    psw

                xmit1:

0209 DB06               in      siostat         ; don't overrun the transmitter if we're
020B E604               ani     xrdymsk         ;  interrupt-driven; wait for TxReady
020D CA0902             jz      xmit1

0210 F1                 pop     psw
                        endif

0211 D304               out     sioxmit         ; blast that byte
0213 C9                 ret


                ; RECV:  Receive a byte from Network A. Set the carry flag if there was
                ;        a receive error.
                ;
                ;        For Z80-SIO receive errors are handled by the special receive
                ;        condition interrupts.

                recv:

                        if      not interrupts
0214 CD5D02             call    netidle
0217 DA2702             jc      rto             ; set error condition if the net went idle

021A DB06               in      siostat         ; else-->wait until a character is in the
021C E601               ani     rrdymsk         ;    buffer
021E CA1402             jz      recv

0221 CD2A02             call    chkstat         ; check for receive errors

                        else
                        ana     a               ; clear carry flag
```



```
                        endif

0224 DB04               in      siorecv         ; input the character
0226 C9                 ret

                rto:                            ; set an error

0227 AF                 xra     a
0228 37                 stc
0229 C9                 ret


                ; CHKSTAT:  Check error status bits of a receive error.  If not error then
                ;           clear the carry flag and return.  Otherwise figure out which
                ;           error occured and increment its counter and set the carry flag.
                ;           Issue an error reset command to the UART.


                chkstat:

022A 3E01               mvi     a,1             ; get error status from SIO read register 1
022C D306               out     siocmd
022E DB06               in      siostat

0230 E670               ani     errbits
0232 C8                 rz                      ; no error occurred-->all done

                        if      netstats        ; gather statistics on the type of error
0233 47                 mov     b,a
0234 E610               ani     pmsk
0236 CA3F02             jz      np              ; not a parity error
```

```
0239 210000             lxi     h,parcntr       ; else-->
023C CD0000             call    inccntr         ; increment parity error counter
                np:
023F 78                 mov     a,b
0240 E605               ani     obit
0242 CA4B02             jz      no              ; not an overrun
0245 210000             lxi     h,ovrcntr       ; else-->
0248 CD0000             call    inccntr         ; increment overrun counter
                no:
024B 78                 mov     a,b
024C E606               ani     fbit
024E CA5702             jz      nf              ; not a framing error
0251 210000             lxi     h,frmcntr       ; else-->
0254 CD0000             call    inccntr         ; increment framing error counter
                nf:
```



```
                        endif
0257 3E30               mvi     a,errst         ; reset error condition
0259 D306               out     siocmd
025B 37                 stc                     ; signal an error
025C C9                 ret
                ; NETIDLE:  See if network A is idle. If idle then set the carry flag.
                netidle:
025D 3E10               mvi     a,10h           ; reset interrupts
025F D306               out     siocmd
0261 D306               out     siocmd          ; do it twice to reject glitches on DCD
0263 DB06               in      siostat         ; is there a data-carrier detect?
0265 E608               ani     carmsk
0267 C8                 rz                      ; yes-->net is in use-->carry flag cleared
0268 AF                 xra     a
0269 CD9A01             call    setbaud         ; net is idle-->reset to hailing rate (9600)
026C 37                 stc                     ; set net idle to true
026D C9                 ret
                        if      interrupts
                ; ENBLRECV:  Enable the channel A receiver interrupts.
                enblrecv:
                        mvi     a,1             ; enable interrupts on all characters
                        out     siocmd
                        mvi     a,011h          ; NOTE: This mask would have to be 015h on
                        out     siocmd          ;  channel B
                        ret
                ; DSBLRECV:  Disable the channel A receiver interrupts.
                dsblrecv:
                        mvi     a,1             ; Disable interrupts on received characters
                        out     siocmd          ;   (Keep status interrupts enabled)
                        out     siocmd          ; NOTE:  Channel B mask is 05h
                        ret
                        endif
```

```
                ; PGMUART:  Program the Network UART channel
                pgmuart:
```



```
                        if      interrupts
                                                ; The 820 already has the SIO vector address
                                                ; programmed from channel B.  Other
                                                ; implementations will have to provide linkage
                                                ; to the vector area in the main XIOS, and
                                                ; load the vector offset into SIO write
                                                ; register 2
                        lxi     h,niisr         ; load status interrupt service routine vector
                        shld    siov5
                        lxi     h,dlisr         ; load transmit ISR vector
                        shld    siov6
                        lxi     h,reisr         ; load receiv ISR vector
                        shld    siov7
                        endif
026E 21D500             lxi     h,sioiblk       ; point to SIO initialization block
0271 060C               mvi     b,sioilen       ; length of block
0273 F3                 di
                pgm1:
0274 7E                 mov     a,m             ; output the block to the SIO
0275 D306               out     siocmd
0277 23                 inx     h
0278 05                 dcr     b
0279 C27402             jnz     pgm1
027C FB                 ei
027D AF                 xra     a               ; set up hailing baud rate = 9600
027E CD9A01             call    setbaud
0281 C9                 ret
                ; INITUART:  Initialize the uart for network A by issuing a reset command
                ;            and clearing out the receive buffer.
                inituart:
0282 3E03               mvi     a,3             ; disable the receiver through register 3
0284 D306               out     siocmd
0286 3EC0               mvi     a,disrcv
0288 D306               out     siocmd
028A DB06               in      siostat         ; is there a garbage byte?
028C E601               ani     rrdymsk
028E CA9602             jz      initu           ; no-->continue initialization
0291 DB04               in      siorecv         ; else-->eat the character
0293 C38202             jmp     inituart        ; try again
                initu:
```



```
0296 3E30               mvi     a,errst         ; reset error conditions
0298 D306               out     siocmd
029A 3E03               mvi     a,3             ; re-enable the receiver
029C D306               out     siocmd
029E 3EC1               mvi     a,enarcv
02A0 D306               out     siocmd
02A2 C9                 ret
                ; INITRECV:  Initialize a receive operation
                initrecv:
```

```
 882
 883   02A3 CD8202            call    inituart
 884
 885                          if      interrupts
 886                          call    enblrecv        ; enable receiver interrupts
 887                          endif
 888
 889   02A6 C9                ret
 890
 891
 892                  ; WAIT - Wait 100 micro seconds
 893
 894                  wait:
 895
 896   02A7 3E16              mvi     a,timeval
 897
 898                  w:
 899
 900   02A9 3D                dcr     a               ; 04
 901   02AA A7                ana     a               ; 04
 902   02AB C2A902            jnz     w               ; 12
 903                                                  ; ---
 904   02AE C9                ret                     ; 30 T-States total
 905
 906
 907                  ; RESTUART:  Reinitialize the UART to the way it was in the
 908                  ;            original BIOS after completing the network operations
 909
 910
 911                  restuart:
 912   02AF C9                ret                     ; UART not used except by network
 913
 914
 915                  ; CSNIOD:  Do any cold start initialization which is necessary.
 916                  ;          Must at least return the value of BAUDS
 917                  ;          If the network uses the printer port then set theh carry flag
 918                  ;          otherwise clear it.
 919
 920                  csniod:
 921
 922   02B0 01032A            lxi     b,bauds         ; return the legal baud rates
```



```
 923   02B3 B7                ora     a               ; not using a printer port
 924   02B4 C9                ret
 925
 926
 927   02B5                   end
```



```
ACTIVE            0010   129#
BAUDGEN           0000    71#  607
BAUDS             2A03    69#  922
BAUDSH            002A    66#   69   623
BAUDSL            0003    65#   69   584
BAUDTBL           00CD   225#  601
BDOS              0005   136#  287
BTBL              00C5   222#  575
BUFLEN            0106   116#
CARBIT            0003    81#
CARMSK            0008    82#  779
CHKSTAT           022A    42   703   725#
CNFGTBLADR        00F6   145   275#
CONFIGTBL         0012   170#  277
CSDLL             0000    53   254
CSNIOD            02B0    43   920#
CURBAUD           00C4   216#  566   610   648
DID               0001   111#  112
DISRCV            00C0    96#  857
DISSLF            004F    98#  597
DLLBAU            00C2    45   210#
DLLON             0000    53   255
DLRECEIVE         0137   357   396#
DLSEND            014A   339   421#
```

```
DSBLXMIT         01F1    44   645#
DTROFF           006A    94#  592   654
DTRON            00EA    93#  635   659
ENARCV           00C1    95#  874
ENASLF           000F    97#  630
ERRBITS          0070    84#  731
ERRST            0030    83#  763   869
FALSE            0000    31#   32    34
FBIT             0006    89#  754
FMSK             0040    90#
FMT              0000   110#  111
FMTBYTE          004B   153#  311
FNC              0003   113#  114
FRMCNTR          0000    55   757
GETRCODE         0000    52   493
GETTCODE         0000    52   455
INCCNTR          0000    55   740   749   758
INITRECV         02A3    42   881#
INITU            0296    40   862   867#
INITUART         0282    41   853#  865   883
INTERRUPTS       0000    34#   47    57   239   511   520   671   695   788   816
                        885
MSG              0005   115#  116
NETADR           00BF    45   209#
NETIDLE          025D    42   696   772#
NETSTATS         FFFF    35#  734
NETWORKERRORMSG  0000   160#  286
NETWORKSTATUS    0012   171#  266
NF               0257   755   760#
NIOS             0000   138#  139
```



```
NO               024B   746   751#
NP               023F   737   742#
NTWRKERROR       00FA   148   283#
NTWRKINIT        00E1   143   252#
NTWRKSTS         00EE   144   264#
NTWRKWBOOT       0103   149   293#
OBIT             0005    87#  745
OMSK             0020    88#
OUTBAU           01C9   603#  639
OVRCNTR          0000    54   748
PARCNTR          0000    54   739
PBIT             0004    85#
PGM1             0274   836#  842
PGMUART          026E    41   814#
PMSK             0010    86#  736
PSRECV           0178   408   487#
PSXMIT           0157   431   449#
PSXRET           0177   468   469   470   471   476#
RBAD             0198   508   509   512   540#
RCVERR           0002   130#  268
RECEIVE          0000    51   489
RECEIVEMSG       0120   147   353#
RECV             0214    40   693#  701
REGSHRT          0000    53   257
RESTUART         02AF    43   911#
RGOOD            0196   507   535#
RQSTRID          0001   152#  174   200   256
RRDYBIT          0000    79#
RRDYMSK          0001    80#  700   861
RRETRY           0139   401#  414
RRTBL            018A   497   505#
RTO              0227   697   712#
RWAIT            017B   491#  517   518   526
SCBAUDT          00CF   230#  638
SELFBIT          0003    91#
SELFCLKD         01D6   581   621#
SELFMSK          0008    92#  580   649
SENDERR          0001   131#  268
SENDMSG          0104   146   305#
SETBAUD          019A    40   563#  783   846
SID              0002   112#  113
SIOCMD           0006    72#  591   593   596   598   629   631   634   636   653
                        655   658   660   728   764   775   776   795   797   805
                        806   839   856   858   870   873   875
```

```
SIOIBLK          00D5   240#  242#  245   832
SIOILEN          000C   245#  833
SIORECV          0004    75#  709   864
SIOSTAT          0006    73#  676   699   729   778   860
SIOV4            FF08   102#
SIOV5            FF0A   103#  825
SIOV6            FF0C   104#  827
SIOV7            FF0E   105#  829
SIOXMIT          0004    74#  683
SIZ              0004   114#  115
```

```
CP/M RMAC ASSEM 1.1     #022    REQUESTER NETWORK I/O SYSTEM FOR ULCNET

SLFCLKD          FFFF    36#  578   589   618   647
TERRCNT          0000    54
TIMEVAL          0016   212#  896
TRANSMIT         0000    51   451
TRETRY           014C   426#  437
TRTBL            0169   458   466#
TRUE             FFFF    32#   35    36
TWAIT            015A   453#  472   473   474
ULCFMT           0000   120#  121
ULCFNC           0004   124#  125
ULCLENHI         0003   123#  124
ULCLENLO         0002   122#  123
ULCMSG           0005   125#
ULCVCIRC         0001   121#  122
W                02A9   898#  902
WAIT             02A7    43   613   894#
XMIT             0208    40   669#
XMIT1            0209   674#  678
XRDYBIT          0002    77#
XRDYMSK          0004    78#  677
```

Listing F-1: Requester Network I/O System for ULCnet

F.3 Creating the ULCnet Server

The server communications software is contained in the modules XIOSNET.ASM and ULCIF.ASM. XIOSNET.ASM contains modifications to MP/M II' s XIOS. ULCIF.ASM is the equivalent of the NETWRKIF transport processes.

ULCIF.ASM uses only two processes, one for input and one for output. To use ULCIF.ASM with the module SERVER.RSP, you must patch SERVER.RSP to write all message responses to a single output queue named NtwrkQO0. This patch is detailed in *CP/NET V1.2 Application Note #2* dated 11-11-82.

The communications interface is interrupt driven, servicing each character as it is received by the network port. ULCIF.ASM requests the network resource through a set of dummy console I/O calls to the XIOS. A call to CONST initializes the network. Calls to CONIN and CONOUT receive and send messages on the network. The communications interface checks network status through a set of poll calls.

The ULCIF input transport process is dispatched at MP/M II cold start. This process makes all necessary queues, creates the ULCIF output process, initializes the network, and writes the configuration table address into the system data page. ULCIF then goes into a loop where it perpetually performs the following actions:

1. Allocates a buffer for an incoming message. If no buffer is available, ULCIF repeats the allocation process until a buffer becomes available.
2. Receives a message by placing the dummy console number in register D, a pointer to the message buffer just allocated in register pair BC, and calling CONIN in the XIOS.
3. Converts the ULCnet format message into CP/NET format. To do this, ULCnet assumes that the virtual circuit number and the requester source ID are identical.

4. Matches the requester ID with a requester control block. If no server is allocated to this requester and the message is a login, ULCIF allocates a server if one is available. Otherwise, ULCIF writes an extended error message to the output queue, NtwrkQO0.
5. Using the requester control block, ULCIF writes the address of the message buffer to the appropriate input queue, NtwrkQI.
6. Repeats.

The output process performs the following actions:

1. Reads the output queue, NtwrkQI0.
2. If the message is a LOGOFF function, frees the appropriate requester control block entry.
3. Converts the message response from CP/NET format into ULCnet format. To do this, ULCnet uses the requester destination ID as the virtual circuit number.
4. Places the dummy console number into register D, the message buffer address into register pair BC, and calls CONOUT in the XIOS.
5. Repeats.

The ULCnet modules DLIF and NIOD are contained in the module XIOSNET.ASM. This module must be incorporated into the server's XIOS. XIOSNET.ASM handles four XIOS jump vector entries, CONST, CONIN, CONOUT, and POLLDEVICE. The jump vector in the XIOS must be modified to point to these routines. XIOSNET contains a linkage to the real XIOS routines for these functions, in this case renamed NCONST, NCONIN, NCONOUT, and POLDEV. The XIOS's interrupt vector might also have to be modified to support the SIO interrupt service routines in IPBMAIN.

When the console I/O routines are entered, they immediately check to see if the dummy console number has been supplied.

Note: you must define a console number that does not conflict with real consoles. Make the dummy console number at least larger than the number of requesters to be supported, since each server process pretends to attach to a unique console ID. If a dummy console number has not been supplied, these routines jump into the real console routines. If the dummy number has been supplied, the routines take the following steps.

CONST:

1. performs network initialization.
2. registers the expected Requester ID's as virtual circuit numbers by repeatedly calling REGSHRT.
3. returns to the ULCIF. This routine is called only once.

CONIN:

1. Calls RECEIVE, using the buffer pointer passed from ULCIF
2. Executes the MP/M II poll function, specifying a poll device routine that repeatedly performs the GETRCODE function until its status shows that a message has been received properly.
3. Returns to the ULCIF.

CONOUT:

1. Calls TRANSMIT, using the buffer pointer passed from ULCIF.
2. Executes the poll function, specifying a poll device routine that repeatedly performs the GETTCODE function until the message has been sent and received by the destination without error.
3. Returns to the ULCIF.

The POLLDEVICE routine behaves almost like the console I/O routines. POLLDEVICE checks for specific poll device numbers to perform network status functions. If these numbers are not detected, control passes to the real POLDEV routine. If network status functions are detected, POLLDEVICE performs the appropriate status check. If the check is successful, a hexadecimal 0FF is returned in register A. If not successful, a 0 is returned.

The MP/M II dispatcher calls POLLDEVICE when it is entered. If the status returned is 0, MP/M II maintains the poll device number on a list and continues to call POLLDEVICE every time it is entered. When the returned status is FF, the dispatcher removes the device number from its list and returns control to the code that originally performed the poll function call, in this case either CONIN or CONOUT. In this manner, the communications interface operates completely transparently, requiring very little CPU resource.

The XIOSNET is designed to be interrupt driven. The IPBMAIN.REL module performs the actual data-link. This module is identical to the IPBMAIN.REL used in the SNIOS. An interrupt-driven protocol is strongly recommended. If you use the polled version, PBMAIN, calls to TRANSMIT and RECEIVE do not return until the requested operation has been performed. This means communications software uses up enormous amounts of CPU time, suspending only when a clock tick interrupts them and forces the dispatcher to be entered. This results in poor server performance.

The interrupt-driven IPBMAIN module sets up the requested operation only when TRANSMIT and RECEIVE are called. The actual protocol is driven by the arrival or departure of each character of the message. This interrupt-driven protocol consumes considerably less CPU time.

To modify the modules ULCIF and XIOSNET for your own server:

1. Patch the module SERVER.RSP to write all of its outputs to a single queue, as described in an application note.
2. Only three parameters must be modified in the ULCIF if four or fewer requesters are to be supported.

   Set NMB$RQSTRS to the number of requesters supported.

   Set NMB$BUFS to the number of requesters, plus one. This extra buffer permits the transmission of LOGIN error messages to the output process, even when all SERVER processes are busy. Having fewer buffers limits the burden on the server at any one time.

   Set CONSOLE$NUM to the dummy console number. The sample listing uses the arbitrarily large number hex 20. This number should be sufficient.
3. If more than four requesters are supported, you must provide extra QCBs, requester control blocks, stack space, and Process Descriptor areas.
4. Modify the XIOS jump vector to jump into the XIOSNET routines CONST, CONIN, CONOUT, and POLLDEVICE. You might have to make additional PUBLIC and EXTRN declarations.
5. Include linkage access to the XIOS interrupt vector. If the XIOS has no interrupt vector, create one.
6. Make sure the false console number specified by the ULCIF module agrees with the one used by XIOSNET.
7. Make sure the device numbers CONIN and CONOUT use in their poll calls do not conflict with other device numbers used by the XIOS.
8. Customize the NIOD section of XIOSNET the same way you customized this section in ULCNIOS.ASM.
9. Create a resident or banked XIOS by linking the regular XIOS module with the network interface:

   ```
   A>LINK RESXI0S=<regu1ar XIOS modu1es>,XI0SNET,IPBMAIN[0S]
   ```

   If you are creating a banked system, all of XIOSNET must reside in common memory.
10. Build the ULCIF.RSP module:

   ```
   A>RMAC ULCIF
   ```

```
A>LINK ULCIF[OR]
```

11. Perform a GENSYS, using the new RESXIOS.SPR, or perform a BNKXIOS.SPR for a banked system. Include the patched SERVER.RSP and ULCIF.RSP modules.

You must have access to the XIOS source modules to implement a ULCnet server in the manner described here. There are two reasons for this:

- Access to the interrupt vector is required.
- Additional device polling routines must be placed into POLLDEVICE.

Both of these problems can be circumvented, but not without difficulty. If the code for XIOSNET is placed in ULCIF, the input process must initialize the interrupt vectors by performing the instruction:

```
LD A,I
```

But to do this, the input process must know where there is empty space in the interrupt page.

Worse is the prospect of not being able to poll for network completion. Instead, the ULCIF might have to drastically reduce its own process priority, then busy wait, making repeated calls to GETTCODE and GETRCODE until the data-link completes. Alternatively, the server can use the polled version of the data-link, PBMAIN.REL. The problems associated with this version have already been described. Placing XIOSNET in the XIOS greatly improves performance.

```
CP/M RMAC ASSEM 1.1     #001    NETWRKIF FOR SYSTEMS RUNNING ULCNET

                                title   'NETWRKIF for Systems Running ULCnet'
                                page    54

                        ;***************************************************************************
                        ;***************************************************************************
                        ;**                                                                       **
                        ;**     S e r v e r   N e t w o r k   I n t e r f a c e   M o d u l e     **
                        ;**                                                                       **
                        ;***************************************************************************
                        ;***************************************************************************


                        ;***************************************************************************
                        ;***************************************************************************
                        ;**                                                                       **
                        ;**     This module performs communication operations on a server         **
                        ;**     equipped with Orange Compuco's ULCnet network adaptor.            **
                        ;**     The actual communications protocol is proprietary to Orange       **
                        ;**     Compuco.  It is included on the CP/NET release disk in REL        **
                        ;**     file format on a module called PBMAIN.REL.  PBMAIN and a data-    **
                        ;**     link interface module, DLIF, must be linked into the XIOS         **
                        ;**     as console I/O routines.  A sample DLIF is included with this     **
                        ;**     module.                                                           **
                        ;**                                                                       **
                        ;**     This module performs the high-level transport and network         **
                        ;**     processing, then calls the DLIF via a direct XIOS console I/O     **
                        ;**     function for data-link.  The following features are supported:   **
                        ;**                                                                       **
                        ;**             o  Queue Minimization using only 2 interface processes    **
                        ;**             o  Dynamic LOGIN/LOGOFF support                           **
                        ;**                                                                       **
                        ;**     Very little of this routine needs to be modified to run an a      **
                        ;**     particular computer system.  The DLIF must be modified to         **
                        ;**     support the system's particular RS-232 hardware, and the XIOS     **
                        ;**     must be modified to support interrupt-driven operation, if so     **
                        ;**     desired, and also support the pseudo-console drivers of the       **
                        ;**     DLIF.                                                             **
                        ;**                                                                       **
                        ;***************************************************************************
```

```
                 ;***************************************************************************

                 ;       This software was developed jointly by
                 ;
                 ;               Digital Research, Inc.
                 ;               P.O. Box 579
                 ;               Pacific Grove, CA 93950
                 ;       and
                 ;               Keybrook Business Systems, Inc.
                 ;               2035 National Avenue
                 ;               Hayward, CA 94545


                 bdosadr:
0000 0000                dw      $-$             ; RSP XDOS entry point
```



```

                 ; User-Configurable Parameters (These should be the only changes needed)

0002 =           nmb$rqstrs      equ     2       ; Number of requesters supported at one time
0003 =           nmb$bufs        equ     3       ; Number of message buffers
0020 =           console$num     equ     20h     ; Pseudo-console number
004B =           fmt$byte        equ     4bh     ; Format byte: short format with acknowledge,
                                                 ;   153.6K baud self-clocked

                 ; Message Buffer Offsets

0000 =           fmt             equ     0               ; format
0001 =           did             equ     fmt+1           ; destination ID
0002 =           sid             equ     did+1           ; source ID
0003 =           fnc             equ     sid+1           ; server function number
0004 =           siz             equ     fnc+1           ; size of message (normalized to 0)
0005 =           msg             equ     siz+1           ; message
0106 =           buf$len         equ     msg+257         ; length of total message buffer

                 ; ULCnet Packet Offsets

0000 =           ulc$fmt         equ     0               ; packet format
0001 =           ulc$v$circ      equ     ulc$fmt+1       ; virtual circuit number
0002 =           ulc$len$lo      equ     ulc$v$circ+1    ; low order of length
0003 =           ulc$len$hi      equ     ulc$len$lo+1    ; high order of length
0004 =           ulc$fnc         equ     ulc$len$hi+1    ; start of message: function code
0005 =           ulc$msg         equ     ulc$fnc+1       ; CP/NET message

                 ; Requester Control Block Offsets

0000 =           rqstr$id        equ     0               ; requester ID for this server
0001 =           uqcb            equ     rqstr$id+1      ; uqcb to queue to this server
0005 =           buf$ptr         equ     uqcb+4          ; queue message <--> msg buffer ptr
0007 =           rcb$len         equ     buf$ptr+2       ; length of requester control block


                 ; NETWRKIF Process Descriptors and Stack Space

                 networkin:                      ; Receiver Process

0002 0000                dw      0               ; link
0004 00                  db      0               ; status
0005 42                  db      66              ; priority
0006 6400                dw      netstkin+46     ; stack pointer
0008 4E45545752          db      'NETWRKIN'      ; name
0010 00                  db      0               ; console
0011 FF                  db      0ffh            ; memseg
0012                     ds      2               ; b
0014                     ds      2               ; thread
0016                     ds      2               ; buff
0018                     ds      1               ; user code & disk slct
0019                     ds      2               ; dcnt
001B                     ds      1               ; searchl
001C                     ds      2               ; searcha
```



```
001E                     ds      2               ; active drives
```

```
0020 0000            dw      0               ; HL'
0022 0000            dw      0               ; DE'
0024 0000            dw      0               ; BC'
0026 0000            dw      0               ; AF'
0028 0000            dw      0               ; IY
002A 0000            dw      0               ; IX
002C 0000            dw      0               ; HL
002E 0000            dw      0               ; DE
0030 0000            dw      0               ; BC
0032 0000            dw      0               ; AF, A = ntwkif console dev #
0034                 ds      2               ; scratch

             netstkin:
0036 C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
003E C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0046 C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
004E C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
0056 C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
005E C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h
0064 B405            dw      setup

             networkout:                     ; Transmitter Process

0066 0000            dw      0               ; link
0068 00              db      0               ; status
0069 42              db      66              ; priority
006A C800            dw      netstkou+46     ; stack pointer
006C 4E45545752      db      'NETWRKOU'      ; name
0074 00              db      0               ; console
0075 FF              db      0ffh            ; memseg
0076                 ds      2               ; b
0078                 ds      2               ; thread
007A                 ds      2               ; buff
007C                 ds      1               ; user code & disk slct
007D                 ds      2               ; dcnt
007F                 ds      1               ; searchl
0080                 ds      2               ; searcha
0082                 ds      2               ; active drives
0084 0000            dw      0               ; HL'
0086 0000            dw      0               ; DE'
0088 0000            dw      0               ; BC'
008A 0000            dw      0               ; AF'
008C 0000            dw      0               ; IY
008E 0000            dw      0               ; IX
0090 0000            dw      0               ; HL
0092 0000            dw      0               ; DE
0094 0000            dw      0               ; BC
0096 0000            dw      0               ; AF, A = ntwkif console dev #
0098                 ds      2               ; scratch

             netstkou:
009A C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
00A2 C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
```



```
00AA C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
00B2 C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
00BA C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h,0c7c7h
00C2 C7C7C7C7        dw      0c7c7h,0c7c7h,0c7c7h
00C8 8606            dw      output


             ; Input queue control blocks

             qcb$in$0:
00CA                 ds      2               ; link
00CC 4E7477726B      db      'NtwrkQI0'      ; name
00D4 0200            dw      2               ; msglen
00D6 0100            dw      1               ; nmbmsgs
00D8                 ds      2               ; dqph
00DA                 ds      2               ; nqph
00DC                 ds      2               ; msgin
00DE                 ds      2               ; msgout
00E0                 ds      2               ; msgcnt
00E2                 ds      2               ; buffer
```

```
                                if      nmb$rqstrs ge 2
                        qcb$in$1:
        00E4                    ds      2               ; link
        00E6 4E7477726B         db      'NtwrkQI1'      ; name
        00EE 0200               dw      2               ; msglen
        00F0 0100               dw      1               ; nmbmsgs
        00F2                    ds      2               ; dqph
        00F4                    ds      2               ; nqph
        00F6                    ds      2               ; msgin
        00F8                    ds      2               ; msgout
        00FA                    ds      2               ; msgcnt
        00FC                    ds      2               ; buffer
                                endif

                                if      nmb$rqstrs ge 3
                        qcb$in$2:
                                ds      2               ; link
                                db      'NtwrkQI2'      ; name
                                dw      2               ; msglen
                                dw      1               ; nmbmsgs
                                ds      2               ; dqph
                                ds      2               ; nqph
                                ds      2               ; msgin
                                ds      2               ; msgout
                                ds      2               ; msgcnt
                                ds      2               ; buffer
                                endif

                                if      nmb$rqstrs ge 4
                        qcb$in$3:
                                ds      2               ; link
                                db      'NtwrkQI3'      ; name
                                dw      2               ; msglen
```



```
                                dw      1               ; nmbmsgs
                                ds      2               ; dqph
                                ds      2               ; nqph
                                ds      2               ; msgin
                                ds      2               ; msgout
                                ds      2               ; msgcnt
                                ds      2               ; buffer
                                endif

                        ; Output queue control blocks

                        qcb$out$0:
        00FE                    ds      2               ; link
        0100 4E7477726B         db      'NtwrkQO0'      ; name
        0108 0200               dw      2               ; msglen
        010A 0300               dw      nmb$bufs        ; nmbmsgs
        010C                    ds      2               ; dqph
        010E                    ds      2               ; nqph
        0110                    ds      2               ; msgin
        0112                    ds      2               ; msgout
        0114                    ds      2               ; msgcnt
        0116                    ds      2*nmb$bufs+1    ; buffer

                        ; Requester Management Table

                        rqstr$table:

                        ;requester 0 control block

        011D FF                 db      0ffh            ; requester ID (marked not in use)
        011E CA00               dw      qcb$in$0        ; UQCB:  QCB pointer
        0120 2201               dw      $+2             ;        pointer to queue message
        0122 0000               dw      $-$             ; pointer to msg buffer (loaded on receive)

                                if      nmb$rqstrs ge 2
                        ;requester 1 control block

        0124 FF                 db      0ffh            ; requester ID (marked not in use)
        0125 E400               dw      qcb$in$1        ; UQCB:  QCB pointer
        0127 2901               dw      $+2             ;        pointer to queue message
        0129 0000               dw      $-$             ; pointer to msg buffer (loaded on receive)
```

```
                        endif

                        if      nmb$rqstrs ge 3
                ;requester 2 control block

                        db      0ffh            ; requester ID (marked not in use)
                        dw      qcb$in$2        ; UQCB:  QCB pointer
                        dw      $+2             ;        pointer to queue message
                        dw      $-$             ; pointer to msg buffer (loaded on receive)
                        endif

                        if      nmb$rqstrs ge 4
                ;requester 3 control block
```



```

                        db      0ffh            ; requester ID (marked not in use)
                        dw      qcb$in$3        ; UQCB:  QCB pointer
                        dw      $+2             ;        pointer to queue message
                        dw      $-$             ; pointer to msg buffer (loaded on receive)
                        endif

                ; Output user queue control block

                uqcb$out$0:
012B FE00               dw      qcb$out$0       ; pointer
012D 2F01               dw      out$buffer$ptr  ; pointer to queue message

                out$buffer$ptr:
012F                    ds      2               ; a queue read will return the message
                                                ; buffer pointer in this location

                ; UQCB for flagging errors from receive process to send process

                uqcb$in$out$0:
0131 FE00               dw      qcb$out$0       ; pointer
0133 3501               dw      in$out$buffer$ptr
                                                ; pointer to queue message

                in$out$buffer$ptr:
0135                    ds      2               ; this pointer used by input process to
                                                ;  to output "server not logged in" errors

                ; Server Configuration Table

                configtbl:
0137 00                 db      0               ; Server status byte
0138 00                 db      0               ; Server processor ID
0139 02                 db      nmb$rqstrs      ; Max number of requesters supported at once
013A 00                 db      0               ; Number of currently logged in requesters
013B 0000               dw      0000h           ; 16 bit vector of logged in requesters
013D                    ds      16              ; Logged In Requester processor ID's
014D 5041535357         db      'PASSWORD'      ; login password

                ; Stacks for server processes.  A pointer to the associated process
                ; descriptor area must reside on the top of each stack.  The stack for
                ; SERVR0PR is internal to SERVER.RSP, and is consequently omitted from the
                ; NETWRKIF module.

0096 =          srvr$stk$len    equ     96h     ; server process stack size

                                if      nmb$rqstrs ge 2
0155            srvr$stk$1:     ds      srvr$stk$len-2
01E9 EB01                       dw      srvr$1$pd
                                endif

                                if      nmb$rqstrs ge 3
                srvr$stk$2:     ds      srvr$stk$len-2
                                dw      srvr$2$pd
```



```
                                endif

                                if      nmb$rqstrs ge 4
```

```
                srvr$stk$3:     ds      srvr$stk$len-2
                                dw      srvr$3$pd
                                endif

                ; Memory allocation for server process descriptor copydown
                ; All server process descriptor allocation must be contiguous

                                if      nmb$rqstrs ge 2
01EB            srvr$1$pd:      ds      52
                                endif

                                if      nmb$rqstrs ge 3
                srvr$2$pd:      ds      52
                                endif

                                if      nmb$rqstrs ge 4
                srvr$3$pd:      ds      52
                                endif


                ; Buffer Control Block: 0 indicates buffer is free for receiving a message
                ; 0ffh indicates that the buffer is in use

                buf$cb:         rept    nmb$bufs
                                db      0
                                endm
021F+00                         DB      0
0220+00                         DB      0
0221+00                         DB      0

                ; Message Buffer Storage Area

                msg$buffers:    rept    nmb$bufs
                                ds      buf$len
                                endm
0222+                           DS      BUF$LEN
0328+                           DS      BUF$LEN
042E+                           DS      BUF$LEN

                ; save area for XIOS routine addresses

                conin$jmp:
0534 C3                 db      jmp
0535 0000       conin:  dw      $-$

                conout$jmp:
0537 C3                 db      jmp
0538 0000       conout: dw      $-$

                constat$jmp:
053A C3                 db      jmp
```



```
                constat:
053B 0000               dw      $-$




                ; NETWRKIF Utility Routines

                ; Operating system linkage routine

                monx:

053D 2A0000             lhld    bdos$adr
0540 E9                 pchl


                ; Double word subtract: DE = HL - DE

                dw$sub:
0541 7D                 mov     a,l
0542 93                 sub     e
0543 5F                 mov     e,a
0544 7C                 mov     a,h
```

```
0545 9A                 sbb     d
0546 57                 mov     d,a
0547 C9                 ret

                ; Routine to scan requester control blocks for a match with the received
                ; source ID.
                ;
                ; Input:  A = Source ID to Match
                ;
                ; Output:
                ;       success:  HL = pointer to requester control block
                ;                 A <> 0FFh
                ;       no match, but a free control block found:
                ;                 HL = pointer to RCB
                ;                  A = 0FFh
                ;                 CY = 0
                ;       no match and no available RCB's:
                ;                  A = 0FFh
                ;                 CY = 1

                scan$table:

0548 211D01             lxi     h,rqstr$table           ;point to the start of the RCB table
054B 0602               mvi     b,nmb$rqstrs
054D 110700             lxi     d,rcb$len               ;size of RCB's for scanning the table

                sc$t1:

0550 BE                 cmp     m                       ;RCB ID = SID?
0551 C8                 rz                              ;yes--> a match--> return
```



```
0552 19                 dad     d                       ;else-->check next entry
0553 05                 dcr     b
0554 C25005             jnz     sc$t1

0557 211D01             lxi     h,rqstr$table           ;no match-->look for a free entry
055A 0602               mvi     b,nmb$rqstrs

                sc$t2:

055C 7E                 mov     a,m
055D 3C                 inr     a
055E CA6A05             jz      sc$t3                   ;an unoccupied entry has been found

0561 19                 dad     d                       ;else-->keep looking
0562 05                 dcr     b
0563 C25C05             jnz     sc$t2

0566 3EFF               mvi     a,0ffh                  ;outa luck-->set the big error
0568 37                 stc
0569 C9                 ret

                sc$t3:                                  ;no match, but found a free entry

056A 3D                 dcr     a                       ;A=0FFh
056B B7                 ora     a                       ;CY=0
056C C9                 ret


                ; This routine free up a requester control block for somebody else who
                ; might want to Log In.
                ;
                ;       Input:  A = source ID that just logged off

                free$rqstr$tbl:

056D 211D01             lxi     h,rqstr$table
0570 110700             lxi     d,rcb$len

                fr$t1:

0573 BE                 cmp     m
0574 C27A05             jnz     fr$t2                   ;RCB ID <> SID-->keep scanning
```

```
0577 36FF               mvi     m,0ffh                  ;else-->mark it as unoccupied
0579 C9                 ret                             ;  and bug out

                fr$t2:

057A 19                 dad     d
057B C37305             jmp     fr$t1                   ;keep going--it's in there somewhere



                ; Routine to send a message on the network
```



```
                ; Input:  HL = pointer to message buffer

                send$msg:

057E E5                 push    h
057F 364B               mvi     m,fmt$byte              ;set ulc$net format byte

0581 23                 inx     h                       ;virtual circuit = requester ID

0582 23                 inx     h
0583 23                 inx     h

0584 46                 mov     b,m                     ;save function number

0585 23                 inx     h                       ;get SIZ
0586 5E                 mov     e,m

0587 1600               mvi     d,0                     ;normalize CP/NET to ULCnet length
0589 13                 inx     d
058A 13                 inx     d

058B 70                 mov     m,b                     ;put FNC in first message byte

058C 2B                 dcx     h                       ;store length
058D 72                 mov     m,d
058E 2B                 dcx     h
058F 73                 mov     m,e

0590 C1                 pop     b                       ;restore buffer pointer
0591 1620               mvi     d,console$num           ;set up fake console number for xios
0593 C33705             jmp     conout$jmp              ;blast that packet


                ; Routine to receive a message on the network
                ; Input:  DE = pointer to buffer

                rcv$message:

0596 42                 mov     b,d
0597 4B                 mov     c,e
0598 C5                 push    b                       ;save buffer pointer
0599 1620               mvi     d,console$num
059B CD3405             call    conin$jmp               ;receive the message

059E E1                 pop     h
059F 3600               mvi     m,0                     ;FMT = 0 (requester to server)

05A1 23                 inx     h
05A2 46                 mov     b,m                     ;save rqstr ID = virtual circuit

05A3 3A3801             lda     configtbl+1
05A6 77                 mov     m,a                     ;DID = server ID

05A7 23                 inx     h
```



```
05A8 5E                 mov     e,m                     ;get low order length

05A9 70                 mov     m,b                     ;SID = requester ID

05AA 23                 inx     h
```

```
05AB 56          mov     d,m                     ;get hi order length
05AC 1B          dcx     d
05AD 1B          dcx     d                       ;normalize ULCnet to CP/NET SIZ
05AE 23          inx     h
05AF 46          mov     b,m                     ;get FNC
05B0 73          mov     m,e                     ;store SIZ
05B1 2B          dcx     h
05B2 70          mov     m,b                     ;store FNC
05B3 C9          ret                             ;ULCnet message formatted
        ; Network I/F Receiver Process
        setup:                                   ;initialize NETWRKIF
05B4 0603        mvi     b,nmb$rqstrs+1          ;loop counter for making n+1 queues
05B6 0E86        mvi     c,134                   ;make queue function code
05B8 11CA00      lxi     d,qcb$in$0
        makeq:                                   ;make all input and output queue(s)
05BB C5          push    b
05BC D5          push    d
05BD CD3D05      call    monx
05C0 E1          pop     h
05C1 111A00      lxi     d,26
05C4 19          dad     d
05C5 EB          xchg
05C6 C1          pop     b
05C7 05          dcr     b
05C8 C2BB05      jnz     makeq
05CB 0E9A        mvi     c,154
05CD CD3D05      call    monx
05D0 110900      lxi     d,9                     ;write configuration table address
05D3 19          dad     d                       ; into system data page, allowing
05D4 113701      lxi     d,configtbl             ;  server initialization to proceed
```



```
05D7 F3          di
05D8 73          mov     m,e
05D9 23          inx     h
05DA 72          mov     m,d
05DB FB          ei
05DC 2B          dcx     h                       ;point to XIOS jump table page
05DD 2B          dcx     h
05DE 2B          dcx     h
05DF 66          mov     h,m
05E0 2E00        mvi     l,0
05E2 110600      lxi     d,6
05E5 19          dad     d                       ;point to constat
05E6 223B05      shld    constat
05E9 23          inx     h
05EA 23          inx     h
05EB 23          inx     h                       ;point to conin
05EC 223505      shld    conin
05EF 23          inx     h
05F0 23          inx     h
05F1 23          inx     h
05F2 223805      shld    conout                  ;point to conout
```

```
05F5 1620               mvi     d,console$num
05F7 CD3A05             call    constat$jmp             ;use constat to initialize ulcnet
05FA 116600             lxi     d,networkout            ;create network I/F output process
05FD 0E90               mvi     c,144
05FF CD3D05             call    monx
                input:                                  ;input process loop
                ; Find a free buffer
0602 211F02             lxi     h,buf$cb                ;point to buffer control block
0605 112202             lxi     d,msg$buffers           ;point to base of buffer area
0608 0603               mvi     b,nmb$bufs              ;get total number of buffers
                input2:
060A 7E                 mov     a,m
060B 3C                 inr     a
060C C22306             jnz     input3                  ;we found a free buffer-->use it
060F E5                 push    h                       ;point to next buffer
0610 210601             lxi     h,buf$len
0613 19                 dad     d
0614 EB                 xchg
0615 E1                 pop     h                       ;point to next buffer control field
0616 23                 inx     h
```



```
0617 05                 dcr     b                       ;have we scanned all the buffers?
0618 C20A06             jnz     input2
061B 0E8E               mvi     c,142                   ;uh oh, we're all clogged up
061D CD3D05             call    monx                    ;dispatch and go sleepy bye for a bit
0620 C30206             jmp     input                   ;try again
                input3:
0623 36FF               mvi     m,0ffh                  ;found a buffer-->mark it used
0625 D5                 push    d
                ; Receive the message
0626 CD9605             call    rcv$message
0629 E1                 pop     h
062A E5                 push    h
062B 23                 inx     h                       ;check requester table to see
062C 23                 inx     h                       ;  whether the source requester
062D 7E                 mov     a,m                     ;    is logged-in
062E CD4805             call    scan$table
0631 3C                 inr     a
0632 CA4A06             jz      input4                  ;not logged-in-->go check for login
                input6:
0635 110500             lxi     d,buf$ptr               ;else-->update message buffer pointer
0638 19                 dad     d
0639 D1                 pop     d
063A 73                 mov     m,e
063B 23                 inx     h
063C 72                 mov     m,d
063D 11FBFF             lxi     d,uqcb-buf$ptr-1        ;point to the uqcb for this requester
0640 19                 dad     d
0641 EB                 xchg
0642 0E8B               mvi     c,139                   ;write the message to the queue
0644 CD3D05             call    monx
```

```
0647 C30206           jmp     input                   ;round and round we go
                input4:                                 ;else-->requester not logged-in
064A D1               pop     d
064B 13               inx     d
064C 13               inx     d
064D 13               inx     d
```



```
064E DA6006           jc      input5                  ;bomb the message if there's no
                                                        ; table entries left
0651 1A               ldax    d
0652 FE40             cpi     64                      ;is it a login?
0654 C26006           jnz     input5
0657 1B               dcx     d                       ;yes-->mark the control block with
0658 1A               ldax    d                       ;  the source ID
0659 77               mov     m,a
065A 1B               dcx     d                       ;go do the queue write
065B 1B               dcx     d
065C D5               push    d
065D C33506           jmp     input6
                input5:                                 ;flag a "not logged in" extended error
0660 EB               xchg
0661 23               inx     h
0662 3601             mvi     m,1                     ;set SIZ=1
0664 23               inx     h
0665 36FF             mvi     m,0ffh                  ;set return code to error
0667 23               inx     h
0668 360C             mvi     m,0ch                   ;flag extended error 12
066A 11FAFF           lxi     d,fmt-msg-1
066D 19               dad     d                       ;point back at message start
066E 3601             mvi     m,1                     ;format = 1
0670 23               inx     h                       ;swap DID and SID
0671 7E               mov     a,m
0672 23               inx     h
0673 46               mov     b,m
0674 77               mov     m,a
0675 2B               dcx     h
0676 70               mov     m,b
0677 2B               dcx     h
0678 223501           shld    in$out$buffer$ptr       ;write buffer pointer to queue msg buf
067B 113101           lxi     d,uqcb$in$out$0         ;write to the queue
067E 0E8B             mvi     c,139
0680 CD3D05           call    monx
0683 C30206           jmp     input                   ;try again
                ;  Network I/F transmitter process
                output:
0686 112B01           lxi     d,uqcb$out$0            ;read the output queue-->go sleepy
0689 0E89             mvi     c,137                   ;  bye until some server process
```



```
068B CD3D05           call    monx                    ;    sends a response
068E 2A2F01           lhld    out$buffer$ptr
0691 EB               xchg
0692 D5               push    d                       ;save message pointer
0693 210300           lxi     h,fnc                   ;get message function code
```

```
   764   0696 19                 dad     d
   765   0697 7E                 mov     a,m
   766   0698 2B                 dcx     h
   767
   768   0699 FE41               cpi     65                      ;is it a logoff?
   769   069B C2A206             jnz     output2
   770
   771   069E 7E                 mov     a,m                     ;load SID
   772   069F CC6D05             cz      free$rqstr$tbl          ;yes-->free up the server process
   773
   774                   output2:
   775
   776   06A2 E1                 pop     h
   777   06A3 E5                 push    h
   778   06A4 CD7E05             call    send$msg                ;send the message
   779
   780   06A7 E1                 pop     h                       ;retrieve message pointer
   781
   782   06A8 112202             lxi     d,msg$buffers           ;DE = pointer - message buffer base
   783   06AB CD4105             call    dw$sub
   784
   785   06AE 011F02             lxi     b,buf$cb                ;BC = DE/buf$len + buf$cb
   786
   787                   output3:
   788
   789   06B1 7B                 mov     a,e
   790   06B2 B2                 ora     d
   791   06B3 CAC106             jz      output4
   792
   793   06B6 EB                 xchg
   794   06B7 110601             lxi     d,buf$len
   795   06BA CD4105             call    dw$sub
   796   06BD 0C                 inr     c
   797   06BE C3B106             jmp     output3
   798
   799                   output4:
   800
   801   06C1 AF                 xra     a
   802   06C2 02                 stax    b                       ;free the buffer for re-use
   803
   804   06C3 C38606             jmp     output                  ;transmission without end, amen
   805
   806   06C6                    end
```

```
CP/M RMAC ASSEM 1.1     #016    NETWRKIF FOR SYSTEMS RUNNING ULCNET

BDOSADR          0000    53#  391
BUFCB            021F   351#  632   785
BUFLEN           0106    72#  361   363   364   365   643   794
BUFPTR           0005    87#   88   680   688
CONFIGTBL        0137   301#  537   594
CONIN            0535   371#  614
CONINJMP         0534   369#  529
CONOUT           0538   375#  619
CONOUTJMP        0537   373#  517
CONSOLENUM       0020    60#  516   528   621
CONSTAT          053B   379#  609
CONSTATJMP       053A   377#  622
DID              0001    67#   68
DWSUB            0541   397#  783   795
FMT              0000    66#   67   729
FMTBYTE          004B    61#  492
FNC              0003    69#   70   763
FREERQSTRTBL     056D   466#  772
FRT1             0573   471#  482
FRT2             057A   474   479#
INOUTBUFFERPTR   0135   292   295#  742
INPUT            0602   628#  655   695   747
INPUT2           060A   636#  651
INPUT3           0623   640   657#
INPUT4           064A   676   697#
INPUT5           0660   703   708   719#
INPUT6           0635   678#  717
MAKEQ            05BB   574#  587
MONX             053D   389#  578   590   626   654   693   746   757
MSG              0005    71#   72   729
MSGBUFFERS       0222   360#  633   782
```

```
NETSTKIN         0036    98   122#
NETSTKOU         009A   136   160#
NETWORKIN        0002    93#
NETWORKOUT       0066   131#  624
NMBBUFS          0003    59#  232   238   351   360   634
NMBRQSTRS        0002    58#  184   198   212   251   260   269   304   317   322
                        327   335   339   343   425   438   570
OUTBUFFERPTR     012F   282   284#  759
OUTPUT           0686   167   753#  804
OUTPUT2          06A2   769   774#
OUTPUT3          06B1   787#  797
OUTPUT4          06C1   791   799#
QCBIN0           00CA   172#  247   572
QCBIN1           00E4   185#  255
QCBOUT0          00FE   228#  281   291
RCBLEN           0007    88#  426   469
RCVMESSAGE       0596   523#  665
RQSTRID          0000    85#   86
RQSTRTABLE       011D   242#  424   437   468
SCANTABLE        0548   422#  673
SCT1             0550   428#  435
SCT2             055C   440#  448
SCT3             056A   444   454#
```

```
CP/M RMAC ASSEM 1.1     #017    NETWRKIF FOR SYSTEMS RUNNING ULCNET

SENDMSG          057E   489#  778
SETUP            05B4   129   568#
SID              0002    68#   69
SIZ              0004    70#   71
SRVR1PD          01EB   319   336#
SRVRSTK1         0155   318#
SRVRSTKLEN       0096   315#  318   323   328
ULCFMT           0000    76#   77
ULCFNC           0004    80#   81
ULCLENHI         0003    79#   80
ULCLENLO         0002    78#   79
ULCMSG           0005    81#
ULCVCIRC         0001    77#   78
UQCB             0001    86#   87   688
UQCBINOUT0       0131   290#  744
UQCBOUT0         012B   280#  755
```

Listing F-2: NETWRKIF for Systems Running ULCnet

```
CP/M RMAC ASSEM 1.1     #001    ULCNET DATA LINK LAYER MP/M XIOS MODULE

                                title 'ULCNET Data Link Layer MP/M XIOS Module'
                                page    54

                        ;*****************************************************************************
                        ;* This module must be linked into the server's XIOS.  It is designed to     *
                        ;* run under MP/M for the Xerox 820, but should be easily customized.  It    *
                        ;* contains the ULCnet interface modules DLIF and NIOD.  The DLIF is an      *
                        ;* interface between the transport software contained in ULCIF.RSP and the   *
                        ;* data-link software contained in IPBMAIN.REL.  The NIOD contains the actual*
                        ;* hardware drivers required to run ULCnet.  The module IPBMAIN.REL must also*
                        ;* be linked into the XIOS.                                                  *
                        ;*****************************************************************************

                        ;       This software is the result of a joint effort between
                        ;
                        ;               Digital Research, Inc.
                        ;               P.O. Box 579
                        ;               Pacific Grove, CA 93950
                        ;       and
                        ;               Keybrook Business Systems, Inc.
                        ;               2035 National Avenue
                        ;               Hayward, CA 94545

                        ; Conditional assembly control

          FFFF =        true            equ     0ffffh
          0000 =        false           equ     not true

```

```
FFFF =          interrupts      equ     true            ; false=polled, true=interrupt-driven
FFFF =          netstats        equ     true            ; switch to gather network statistics
FFFF =          slfclkd         equ     true            ; supports self-clocked operation

                ; Linkage information

                        public  nconst,nconin,nconout   ; XIOS console jump table entries
                        public  polldevice              ; XIOS polling routine
                        public  setbaud,xmit,recv,initu ; NIOD routines called by IPBMAIN
                        public  inituart,pgmuart
                        public  chkstat,netidle,initrecv
                        public  wait,restuart,csniod
                        public  dsblxmit
                        public  dllbau,netadr

                        if      interrupts
                        public  enblrecv,dsblrecv
                        endif

                        extrn   transmit,receive        ; IPBMAIN routines and objects
                        extrn   gettcode,getrcode
                        extrn   csdll,dllon,regshrt
                        extrn   terrcnt,parcntr,ovrcntr
                        extrn   frmcntr,inccntr
                        extrn   xdos,const,conin,conout ; linkage back to the rest of XIOS
                        extrn   poldev
```



```

                        if      interrupts
                        extrn   rtmochk                 ; IPBMAIN interrupt routines
                        extrn   dlisr,reisr,niisr
                        endif


                ; Hardware definitions for the Z80-SIO channel A - For the Xerox 820.

0003 =          baudsl  equ     03h             ; Usable baud rates: 9600, 19.2K asynch.,
002A =          baudsh  equ     2ah             ; 76.8K, 153.6K, 307.2K self-clocked

                                                ; baud rate capability mask
2A03 =          bauds   equ     (baudsh*100h)+baudsl

0000 =          baudgen equ     0               ; External baud rate generator register
0006 =          siocmd  equ     6               ; Command/Mode register
0006 =          siostat equ     6               ; Status register
0004 =          sioxmit equ     4               ; Transmit register
0004 =          siorecv equ     4               ; Receive register

0002 =          xrdybit equ     2               ; Transmit buffer empty status bit
0004 =          xrdymsk equ     4               ; transmit buffer empty status mask
0000 =          rrdybit equ     0               ; Receive buffer full status bit
0001 =          rrdymsk equ     1               ; receive buffer full status mask
0003 =          carbit  equ     3               ; Net Idle detect bit position
0008 =          carmsk  equ     8               ; Net Idle detect mask
0030 =          errst   equ     030h            ; Error flag reset
0070 =          errbits equ     070h            ; Error bit position mask
0004 =          pbit    equ     4               ; Parity error bit position
0010 =          pmsk    equ     10h             ; parity error mask
0005 =          obit    equ     5               ; Overrun error bit position
0020 =          omsk    equ     20h             ; overrun error mask
0006 =          fbit    equ     6               ; Framing error bit position
0040 =          fmsk    equ     40h             ; framing error mask
0003 =          selfbit equ     3               ; Self clock bit position
0008 =          selfmsk equ     8               ; slef clock bit mask
00EA =          dtron   equ     0eah            ; Turn on DTR
006A =          dtroff  equ     06ah            ; Turn off DTR
00C1 =          enarcv  equ     0c1h            ; Enable receive-clock
00C0 =          disrcv  equ     0c0h            ; Disable receive clock
000F =          enaslf  equ     00fh            ; Enable Self-clock mode
004F =          disslf  equ     04fh            ; Disable Self-clock mode

                ; SIO Mode 2 interrupts vector table

FF08 =          siov4   equ     0ff08h          ; SIO port A xmit buffer empty
FF0A =          siov5   equ     0ff0ah          ; SIO port A external status change
```

```
        FF0C =         siov6   equ     0ff0ch          ; SIO port A receive
        FF0E =         siov7   equ     0ff0eh          ; SIO port A special receive condition

        0020 =         netcon  equ     20h             ; fake console number called by ULCIF for
                                                       ; network operations

```



```
                       ; polling equates

        0020 =         ulctx   equ     20h             ; transmission poll number
        0021 =         ulcrx   equ     21h             ; receive poll number
                               page
```



```



                       ; ULCnet Data Definitions

        0000           netadr: ds      3               ;ULCnet network address
        0003           dllbau: ds      2               ;baud rate mask

        0016 =         timeval equ     22              ; WAIT routine time constant
                                                       ; 12 for 2.5 megahertz Z80
                                                       ; 22 for 4.0 megahertz Z80

                       dev$table:                      ;polling device table

        0005 9800              dw      twait           ;receive poll wait
        0007 D300              dw      rwait           ;transmit poll wait
        0002 =         num$devices     equ     ($-dev$table)/2

        0009           tcode:  ds      1                       ; Transmit Return code
        000A           rcode:  ds      1                       ; Receive Return code

        000B FF        curbaud db      0ffh                    ; Current baud rate


        000C 0102040810btbl:   db      1,2,4,8,16,32,64,128    ; table to convert baud number codes
                                                               ;   into a bit mask

                       baudtbl:                                ; async baud rate table

        0014 0E                db      0eh                     ; 9600 Baud
        0015 0F                db      0fh                     ; 19200

                       scbaudt:                                ; self-clock baud rate table

        0016 00                db      0                       ;  62500 Baud - Not implemented
        0017 0D                db      0dh                     ;  76800 Baud
        0018 00                db      0                       ; 125000 Baud - Not implemented
        0019 0E                db      0eh                     ; 153600 Baud
        001A 00                db      0                       ; 250000 Baud - Not implemented
        001B 0F                db      0fh                     ; 307200 Baud

                               if      interrupts
        001C 30144F156Asioiblk db      030h,14h,4fh,15h,06ah,13h,0c1h,11h,01h,10h,10h,30h
                               else
                       sioiblk db      030h,14h,4fh,15h,06ah,13h,0c1h,11h,00h,10h,10h,30h
                               endif

        000C =         sioilen equ     $-sioiblk

                               page
```



```



                       ; ULCnet data-link interface code
```

```


                ; POLLDEVICE:  Device polling routine.
                ;       Input:
                ;               C = device number to poll
                ;       Output:
                ;               A = 0    if not ready
                ;                   0ffh if ready

                polldevice:

0028 79                 mov     a,c             ; if not a network poll, go to the real
0029 D620               sui     ulctx           ;   routine
002B DA0000             jc      poldev

002E FE02               cpi     num$devices     ; check for poll number in bounds
0030 DA3600             jc      devok

0033 3E00               mvi     a,0             ; out-of-bounds-->don't do anything
0035 C9                 ret

                devok:

0036 6F                 mov     l,a
0037 2600               mvi     h,0
0039 29                 dad     h               ; multiply index by 2

003A 110500             lxi     d,dev$table     ; index into the poll routine table
003D 19                 dad     d

003E 5E                 mov     e,m
003F 23                 inx     h
0040 56                 mov     d,m             ; get the routine address

0041 EB                 xchg
0042 E9                 pchl                    ; dispatch



                ;
                ; NCONST: Console status entry point.  If register D = fake network
                ;         console ID, do network initialization.  Otherwise, go back to
                ;         the real console routines.

                nconst:

0043 3E20               mvi     a,netcon        ; Check if network call
0045 BA                 cmp     d
0046 C20000             jnz     const           ; Jump to normal CONST if not network

```



```
0049 CD0000             call    csdll           ; Cold start the data link
004C CD0000             call    dllon           ; Initialize the SIO Drivers
004F AF                 xra     a               ; Initialize all the short addresses

                nxtadd:

0050 3C                 inr     a
0051 FE05               cpi     5               ; Check for last address
0053 C8                 rz
0054 F5                 push    psw
0055 CD0000             call    regshrt
0058 F1                 pop     psw
0059 C35000             jmp     nxtadd          ; Jump to process next address


                ; NCONIN:  Console In entry point.  If register D = the fake network ID
                ;          then receive a network message, using polled status checks of
                ;          an interrupt-driven data-link.  Otherwise, go back to the real
                ;          CONIN routine.

                nconin:

005C 3E20               mvi     a,netcon        ; Check for network call
005E BA                 cmp     d
```

```
005F C20000          jnz     conin           ; Jump to normal CONIN if not network

0062 50              mov     d,b             ; Setup for PSRECEIVE
0063 59              mov     e,c

                rretry:

0064 AF              xra     a               ; Packet mode
0065 010101          lxi     b,257           ; Buffer size
0068 210000          lxi     h,0             ; Infinite wait
006B D5              push    d               ; Save buffer address for retry
006C CDC100          call    psrecv
006F D1              pop     d               ; Restore buffer address
0070 B7              ora     a
0071 C8              rz                      ; Return if no error

0072 C36400          jmp     rretry          ; Jump to try again if error


                ; NCONOUT:  Console out entry point.  If D = fake console ID, send a network
                ;           message.  Otherwise, just head for the real CONOUT routine.


                nconout:

0075 3E20            mvi     a,netcon        ; Check for network call
0077 BA              cmp     d
0078 C20000          jnz     conout          ; Jump to normal CONOUT if not network

007B 50              mov     d,b             ; Setup for PSXMIT
```



```
007C 59              mov     e,c

                tretry:

007D AF              xra     a               ; Packet mode, wait for Net Idle
007E D5              push    d               ; Save buffer address for retry
007F CD8800          call    psxmit
0082 D1              pop     d               ; Restore buffer address
0083 B7              ora     a
0084 C8              rz                      ; Return if no error

0085 C37D00          jmp     tretry          ; Jump to retry if error


                ; PSXMIT:  Transmit the packet pointed at by DE.  If carry flag is set
                ;          then don't wait for the Net to become idle.
                ;
                ; Returns the completion code in A:
                ;
                ;       0       - Transmission ok and Data Link Ack Received
                ;                 (In the case of multicast, no Ack required)
                ;       2       - Transmission OK but no Data Link Ack received.
                ;
                ;       4       - Other error.

                psxmit:

0088 CD0000          call    transmit                ; TRETCODE := TRANSMIT(TBUFPTR,)

008B 0E83            mvi     c,83h                   ; Poll the transmitter for completion
008D 1E20            mvi     e,ulctx
008F CD0000          call    xdos

0092 3A0900          lda     tcode                   ; Fetch return code
0095 C3CE00          jmp     exitdl

                ; TWAIT:  Transmission completion poll routine.
                ;
                ;         Output:
                ;               A = 0    if not complete
                ;                   0ffh if complete

                twait:

```

```
0098 CD0000           call    gettcode                ; A := GETTCODE - Xmit return code

009B 5F               mov     e,a                     ; get return code processing vectore
009C 1600             mvi     d,0
009E 21A700           lxi     h,trtbl
00A1 19               dad     d

00A2 5E               mov     e,m                     ; dispatch on return code
00A3 23               inx     h
00A4 66               mov     h,m
```



```
00A5 6B               mov     l,e
00A6 E9               pchl

              ; Return code dispatch table

00A7 B700     trtbl:  dw      psxret                  ; Good transmission
00A9 B700             dw      psxret                  ; No Data Link Ack
00AB B700             dw      psxret                  ; Too many collisions
00AD B700             dw      psxret                  ; Transmitter is disabled
00AF B500             dw      tsleep                  ; Transmitter is idle
00B1 B500             dw      tsleep                  ; Transmitter is in progress
00B3 B500             dw      tsleep                  ; Transmitter is waiting for ack

              tsleep:

00B5 AF               xra     a                       ; Code for continue to sleep
00B6 C9               ret

              psxret:                                 ; Enter here if something happened

00B7 D2BB00           jnc     twakeup                 ; Jump if no transmit error
00BA 2F               cma                             ; Else-->Indicate error

              twakeup:

00BB 320900           sta     tcode                   ; Store return code
00BE 3EFF             mvi     a,0ffh                  ; Signal poll successful
00C0 C9               ret



              ; PSRECV:  Receive a packet into buffer pointed at by DE.  Length of
              ;          packet must be less than length of buffer in BC. HL is the receive
              ;          timeout count.
              ;
              ;          Upon return clear the carry bit if a packet received and ACKed.
              ;          Set the carry flag if any error occured.
              ;


              psrecv:

00C1 CD0000           call    receive                 ;  := RECEIVE(HL,DE,BC)

00C4 0E83             mvi     c,83h                   ; Poll until receive complete
00C6 1E21             mvi     e,ulcrx
00C8 CD0000           call    xdos

00CB 3A0A00           lda     rcode                   ; Fetch return code

              ; Common exit routine for returning to the pseudo-console handler

              exitdl:

```



```
00CE B7               ora     a                       ; Assume no error
00CF F0               rp                              ; Return if no error

00D0 2F               cma
00D1 37               stc                             ; Indicate error
00D2 C9               ret
```

```
                ; RWAIT:  Poll routine to detect receive status.
                ;
                ;       Output:
                ;               A = 0    if receive not complete
                ;                   0ffh if receive complete

                rwait:

00D3 CD0000             call    getrcode                ; A := GETRCODE

00D6 5F                 mov     e,a                     ; form dispatch vector
00D7 1600               mvi     d,0
00D9 21E200             lxi     h,rrtbl
00DC 19                 dad     d

00DD 5E                 mov     e,m                     ; dispatch on receive completion code
00DE 23                 inx     h
00DF 66                 mov     h,m
00E0 6B                 mov     l,e
00E1 E9                 pchl

                ; Receive completion code dispatch table

00E2 F000       rrtbl:  dw      rgood                   ; Good receive
00E4 F600               dw      rbad                    ; Bad receive
00E6 F600               dw      rbad                    ; Disabled

                        if      not interrupts
                        dw      rbad                    ; Still idle after timeout
                        else
00E8 FA00               dw      ridle                   ; Idle
                        endif

00EA EE00               dw      rsleep                  ; Inprogress
00EC EE00               dw      rsleep                  ; In progress and for us.

                rsleep:

00EE AF                 xra     a                       ; Code for continue to sleep
00EF C9                 ret

                rgood:
                rwakeup:

00F0 320A00             sta     rcode                   ; Store return code
00F3 3EFF               mvi     a,0ffh                  ; Wake up code
00F5 C9                 ret
```



```
                rbad:

00F6 2F                 cma                             ; Code for error
00F7 C3F000             jmp     rwakeup                 ; Jump to wake up receive process

                        if interrupts

                ridle:

00FA CD0000             call    rtmochk                 ; Check for timeout
00FD DAF600             jc      rbad                    ; if timeout, signal error
0100 C3EE00             jmp     rsleep                  ; Continue to wait if no timeout

0103 C9                 ret

                        endif
                        page
```



```
                ; NIOD routines
```

```
                  ; SETBAUD:  Set the baud rate based on the baud rate code in A.  Do special
                  ;           logic for self-clocked mode.
                  ;
                  ;               0 = 9600 baud
                  ;               1 = 19200 baud
                  ;               9 = 76800 baud self-clock
                  ;               11= 153600 baud self-clock
                  ;               13= 307200 baud self-clock
                  ;
                  ; If this station cannot handle the requested baud rate, then set
                  ; the carry flag.

                  setbaud:

0104 E60F                 ani     0fh             ; mask all but the baud bits
0106 210B00               lxi     h,curbaud       ; are we at the current baud rate?
0109 BE                   cmp     m
010A C8                   rz                      ; yes-->all done

010B 47                   mov     b,a             ; else-->get baud rate generator value
010C E607                 ani     7
010E 5F                   mov     e,a
010F 1600                 mvi     d,0

0111 210C00               lxi     h,btbl          ; point to vertical-to-horizontal decode
0114 19                   dad     d               ;   table

                          if      slfclkd
0115 78                   mov     a,b
0116 E608                 ani     selfmsk         ; is this a self-clocked value?
0118 C24001               jnz     selfclkd
                          endif

011B 3E03                 mvi     a,baudsl        ; get legal baud rate mask
011D A6                   ana     m
011E 37                   stc
011F C8                   rz                      ; return with error if its an illegal rate

                          if      slfclkd
0120 3E05                 mvi     a,5             ; else-->switch off possible self-clock mode
0122 D306                 out     siocmd
0124 3E6A                 mvi     a,dtroff        ; disable DTR in SIO register 5
0126 D306                 out     siocmd

0128 3E04                 mvi     a,4             ; disable sync mode in register 4
012A D306                 out     siocmd
012C 3E4F                 mvi     a,disslf
012E D306                 out     siocmd
```



```
                          endif

0130 211400               lxi     h,baudtbl       ; point to async baud rate table

                  outbau:

0133 19                   dad     d               ; get async baud rate value
0134 7E                   mov     a,m
0135 D300                 out     baudgen         ; load it into the baud rate generator
                                                  ; NOTE: This is not a CTC

0137 210B00               lxi     h,curbaud
013A 70                   mov     m,b             ; set current baud byte

013B CD1E02               call    wait            ; allow the system to reach equilibrium

013E A7                   ana     a               ; return success
013F C9                   ret

                          if      slfclkd
                  ; Throw SIO into self-clocked mode

                  selfclkd:
```

```
0140 3E2A          mvi     a,baudsh        ; Is this a legal rate?
0142 A6            ana     m
0143 37            stc
0144 C8            rz                      ; return an error if not
0145 3E04          mvi     a,4             ; enable sync mode in register 4
0147 D306          out     siocmd
0149 3E0F          mvi     a,enaslf
014B D306          out     siocmd
014D 3E05          mvi     a,5             ; enable DTR in register 5
014F D306          out     siocmd
0151 3EEA          mvi     a,dtron
0153 D306          out     siocmd
0155 211600        lxi     h,scbaudt       ; point to baud rate table for self-clock mode
0158 C33301        jmp     outbau          ; program the baud rate generator
                   endif
           ; DSBLXMIT:  Disable the transmitter if in self clocked mode
           dsblxmit:
                   if      slfclkd
015B 3A0B00        lda     curbaud         ; are we in self-clocked mode?
015E E608          ani     selfmsk
0160 C8            rz                      ; no-->don't bother
0161 3E05          mvi     a,5             ; disable SIO from transmitting by disabling
```



```
0163 D306          out     siocmd          ; DTR in register 5
0165 3E6A          mvi     a,dtroff
0167 D306          out     siocmd
0169 3E05          mvi     a,5             ; Enable receive by re-enabling DTR
016B D306          out     siocmd
016D 3EEA          mvi     a,dtron
016F D306          out     siocmd
                   endif
0171 C9            ret
           ; XMIT:  Transmit the byte in A on network A.
           xmit:
                   if      not interrupts
                   push    psw
           xmit1:
                   in      siostat         ; don't overrun the transmitter if we're
                   ani     xrdymsk         ;  interrupt-driven; wait for TxReady
                   jz      xmit1
                   pop     psw
                   endif
0172 D304          out     sioxmit         ; blast that byte
0174 C9            ret
           ; RECV:  Receive a byte from Network A. Set the carry flag if there was
           ;        a receive error.
           ;
           ;        For Z80-SIO receive errors are handled by the special receive
           ;        condition interrupts.
           recv:
                   if      not interrupts
                   call    netidle
```

```
                              jc      rto             ; set error condition if the net went idle
                              in      siostat         ; else-->wait until a character is in the
                              ani     rrdymsk         ;    buffer
                              jz      recv
                              call    chkstat         ; check for receive errors
                              else
      0175 A7                 ana     a               ; clear carry flag
```



```
                              endif
      0176 DB04               in      siorecv         ; input the character
      0178 C9                 ret
                      rto:                            ; set an error
      0179 AF                 xra     a
      017A 37                 stc
      017B C9                 ret
                      ; CHKSTAT:  Check error status bits of a receive error.  If not error then
                      ;           clear the carry flag and return.  Otherwise figure out which
                      ;           error occured and increment its counter and set the carry flag.
                      ;           Issue an error reset command to the UART.
                      chkstat:
      017C 3E01               mvi     a,1             ; get error status from SIO read register 1
      017E D306               out     siocmd
      0180 DB06               in      siostat
      0182 E670               ani     errbits
      0184 C8                 rz                      ; no error occurred-->all done
                              if      netstats        ; gather statistics on the type of error
      0185 47                 mov     b,a
      0186 E610               ani     pmsk
      0188 CA9101             jz      np              ; not a parity error
      018B 210000             lxi     h,parcntr       ; else-->
      018E CD0000             call    inccntr         ; increment parity error counter
                      np:
      0191 78                 mov     a,b
      0192 E605               ani     obit
      0194 CA9D01             jz      no              ; not an overrun
      0197 210000             lxi     h,ovrcntr       ; else-->
      019A CD0000             call    inccntr         ; increment overrun counter
                      no:
      019D 78                 mov     a,b
      019E E606               ani     fbit
      01A0 CAA901             jz      nf              ; not a framing error
      01A3 210000             lxi     h,frmcntr       ; else-->
      01A6 CD0000             call    inccntr         ; increment framing error counter
                      nf:
```



```
                              endif
      01A9 3E30               mvi     a,errst         ; reset error condition
      01AB D306               out     siocmd
      01AD 37                 stc                     ; signal an error
      01AE C9                 ret
```

```
                ; NETIDLE:  See if network A is idle. If idle then set the carry flag.

                netidle:

01AF 3E10               mvi     a,10h           ; reset interrupts
01B1 D306               out     siocmd
01B3 D306               out     siocmd          ; do it twice to reject glitches on DCD

01B5 DB06               in      siostat         ; is there a data-carrier detect?
01B7 E608               ani     carmsk
01B9 C8                 rz                      ; yes-->net is in use-->carry flag cleared

01BA AF                 xra     a
01BB CD0401             call    setbaud         ; net is idle-->reset to hailing rate (9600)
01BE 37                 stc                     ; set net idle to true
01BF C9                 ret


                        if      interrupts

                ; ENBLRECV:  Enable the channel A receiver interrupts.

                enblrecv:

01C0 3E01               mvi     a,1             ; enable interrupts on all characters
01C2 D306               out     siocmd
01C4 3E11               mvi     a,011h          ; NOTE: This mask would have to be 015h on
01C6 D306               out     siocmd          ;  channel B
01C8 C9                 ret

                ; DSBLRECV:  Disable the channel A receiver interrupts.

                dsblrecv:

01C9 3E01               mvi     a,1             ; Disable interrupts on received characters
01CB D306               out     siocmd          ;   (Keep status interrupts enabled)
01CD D306               out     siocmd          ; NOTE:  Channel B mask is 05h
01CF C9                 ret

                        endif


                ; PGMUART:  Program the Network UART channel

                pgmuart:
```



```
                        if      interrupts
                                                ; The 820 already has the SIO vector address
                                                ; programmed from channel B.  Other
                                                ; implementations will have to provide linkage
                                                ; to the vector area in the main XIOS, and
                                                ; load the vector offset into SIO write
                                                ; register 2

01D0 210000             lxi     h,niisr         ; load status interrupt service routine vector
01D3 220AFF             shld    siov5
01D6 210000             lxi     h,dlisr         ; load transmit ISR vector
01D9 220CFF             shld    siov6
01DC 210000             lxi     h,reisr         ; load receiv ISR vector
01DF 220EFF             shld    siov7
                        endif

01E2 211C00             lxi     h,sioiblk       ; point to SIO initialization block
01E5 060C               mvi     b,sioilen       ; length of block
01E7 F3                 di

                pgm1:

01E8 7E                 mov     a,m             ; output the block to the SIO
01E9 D306               out     siocmd
01EB 23                 inx     h
```

```
01EC 05              dcr     b
01ED C2E801          jnz     pgm1

01F0 FB              ei
01F1 AF              xra     a               ; set up hailing baud rate = 9600
01F2 CD0401          call    setbaud
01F5 C9              ret


             ; INITUART:  Initialize the uart for network A by issuing a reset command
             ;            and clearing out the receive buffer.

             inituart:

01F6 3E03            mvi     a,3             ; disable the receiver through register 3
01F8 D306            out     siocmd
01FA 3EC0            mvi     a,disrcv
01FC D306            out     siocmd

01FE DB06            in      siostat         ; is there a garbage byte?
0200 E601            ani     rrdymsk
0202 CA0A02          jz      initu           ; no-->continue initialization

0205 DB04            in      siorecv         ; else-->eat the character
0207 C3F601          jmp     inituart        ; try again

             initu:

```



```
020A 3E30            mvi     a,errst         ; reset error conditions
020C D306            out     siocmd

020E 3E03            mvi     a,3             ; re-enable the receiver
0210 D306            out     siocmd
0212 3EC1            mvi     a,enarcv
0214 D306            out     siocmd

0216 C9              ret

             ; INITRECV:  Initialize a receive operation

             initrecv:

0217 CDF601          call    inituart

                     if      interrupts
021A CDC001          call    enblrecv        ; enable receiver interrupts
                     endif

021D C9              ret


             ; WAIT - Wait 100 micro seconds

             wait:

021E 3E16            mvi     a,timeval

             w:

0220 3D              dcr     a               ; 04
0221 A7              ana     a               ; 04
0222 C22002          jnz     w               ; 12
                                             ; ---
0225 C9              ret                     ; 30 T-States total


             ; RESTUART:  Reinitialize the UART to the way it was in the
             ;            original BIOS after completing the network operations


             restuart:
0226 C9              ret                     ; UART not used except by network

```

```
   822                  ; CSNIOD:  Do any cold start initialization which is necessary.
   823                  ;          Must at least return the value of BAUDS
   824                  ;          If the network uses the printer port then set theh carry flag
   825                  ;          otherwise clear it.
   826
   827                  csniod:
   828
   829   0227 01032A            lxi     b,bauds         ; return the legal baud rates
```



```
   830   022A B7                ora     a               ; not using a printer port
   831   022B C9                ret
   832
   833   022C                   end
```



```
BAUDGEN          0000    70#  514
BAUDS            2A03    68#  829
BAUDSH           002A    65#   68   530
BAUDSL           0003    64#   68   491
BAUDTBL          0014   141#  508
BTBL             000C   138#  482
CARBIT           0003    80#
CARMSK           0008    81#  686
CHKSTAT          017C    39   610   632#
CONIN            0000    53   242
CONOUT           0000    53   269
CONST            0000    53   216
CSDLL            0000    50   218
CSNIOD           0227    40   827#
CURBAUD          000B   135#  473   517   555
DEVOK            0036   184   189#
DEVTABLE         0005   126#  130   195
DISRCV           00C0    95#  764
DISSLF           004F    97#  504
DLISR            0000    58   733
DLLBAU           0003    42   120#
DLLON            0000    50   219
DSBLRECV         01C9    45   709#
DSBLXMIT         015B    41   552#
DTROFF           006A    93#  499   561
DTRON            00EA    92#  542   566
ENARCV           00C1    94#  781
ENASLF           000F    96#  537
ENBLRECV         01C0    45   699#  793
ERRBITS          0070    83#  638
ERRST            0030    82#  670   776
EXITDL           00CE   306   378#
FALSE            0000    27#
FBIT             0006    88#  661
FMSK             0040    89#
FRMCNTR          0000    52   664
GETRCODE         0000    49   395
GETTCODE         0000    49   316
INCCNTR          0000    52   647   656   665
INITRECV         0217    39   788#
INITU            020A    37   769   774#
INITUART         01F6    38   760#  772   790
INTERRUPTS       FFFF    29#   44    56   155   414   440   578   602   695   723
                        792
NCONIN           005C    35   238#
NCONOUT          0075    35   265#
NCONST           0043    35   212#
NETADR           0000    42   119#
NETCON           0020   106#  214   240   267
NETIDLE          01AF    39   603   679#
NETSTATS         FFFF    30#  641
NF               01A9   662   667#
NIISR            0000    58   731
NO               019D   653   658#
```

```
NP              0191   644   649#
NUMDEVICES      0002   130#  183
NXTADD          0050   222#  230
OBIT            0005    86#  652
OMSK            0020    87#
OUTBAU          0133   510#  546
OVRCNTR         0000    51   655
PARCNTR         0000    51   646
PBIT            0004    84#
PGM1            01E8   743#  749
PGMUART         01D0    38   721#
PMSK            0010    85#  643
POLDEV          0000    54   181
POLLDEVICE      0028    36   177#
PSRECV          00C1   253   366#
PSXMIT          0088   278   297#
PSXRET          00B7   331   332   333   334   344#
RBAD            00F6   411   412   415   435#  445
RCODE           000A   133#  374   431
RECEIVE         0000    48   368
RECV            0175    37   600#  608
REGSHRT         0000    50   228
REISR           0000    58   735
RESTUART        0226    40   818#
RGOOD           00F0   410   428#
RIDLE           00FA   417   442#
RRDYBIT         0000    78#
RRDYMSK         0001    79#  607   768
RRETRY          0064   247#  258
RRTBL           00E2   399   410#
RSLEEP          00EE   420   421   423#  446
RTMOCHK         0000    57   444
RTO             0179   604   619#
RWAIT           00D3   129   393#
RWAKEUP         00F0   429#  438
SCBAUDT         0016   146#  545
SELFBIT         0003    90#
SELFCLKD        0140   488   528#
SELFMSK         0008    91#  487   556
SETBAUD         0104    37   470#  690   753
SIOCMD          0006    71#  498   500   503   505   536   538   541   543   560
                       562   565   567   635   671   682   683   702   704   712
                       713   746   763   765   777   780   782
SIOIBLK         001C   156#  158#  161   739
SIOILEN         000C   161#  740
SIORECV         0004    74#  616   771
SIOSTAT         0006    72#  583   606   636   685   767
SIOV4           FF08   101#
SIOV5           FF0A   102#  732
SIOV6           FF0C   103#  734
SIOV7           FF0E   104#  736
SIOXMIT         0004    73#  590
SLFCLKD         FFFF    31#  485   496   525   554
TCODE           0009   132#  305   351
```



```
TERRCNT         0000    51
TIMEVAL         0016   122#  803
TRANSMIT        0000    48   299
TRETRY          007D   274#  283
TRTBL           00A7   320   331#
TRUE            FFFF    26#   27    29    30    31
TSLEEP          00B5   335   336   337   339#
TWAIT           0098   128   314#
TWAKEUP         00BB   346   349#
ULCRX           0021   112#  371
ULCTX           0020   111#  180   302
W               0220   805#  809
WAIT            021E    40   520   801#
XDOS            0000    53   303   372
XMIT            0172    37   576#
XRDYBIT         0002    76#
XRDYMSK         0004    77#  584
```

Listing F-3: ULCnet Data-link Layer MP/M XIOS Module

Appendix G
Using CP/NET 1.2 with CORVUS OMNINET

Corvus OMNINET is an inexpensive, high-performance CSMA/CA networking system supporting up to 63 hosts on a one-megabit-per second, twisted-pair cable. OMNINET host interface adaptors are intelligent coprocessors that deal with all aspects of network communication of the host in which they are installed, up to and including the transport layer of the ISO open system model. The sample SNIOS and NETWRKIF files following this discussion show one way to use Corvus engineering transporters to implement a CP/NET system.

G.1 The Corvus Engineering Transporter

The Corvus engineering transporter is a card for evaluating Corvus OMNINET with minimum modification to an existing Z80 system. The transporter is not an end-user product, but it is similar enough in hardware design to most production systems using OMNINET to work with little modification.

General information about the Corvus transporter is presented here to help you understand the operation of the sample codes at the end of this appendix. For more information, refer to Corvus documentation.

Communication with the transporter hardware is simplified by the fact that the transporter is microprocessor-based and uses autonomous DMA to access its host computer's memory directly. All communication between host and transporter is controlled by well organized data structures existing in host memory. The only port I/O the host ever does is the transmission, to the transporter hardware, of 24-bit pointer objects (as three serial bytes, most significant byte first) via an output port. Note that all Corvus multibyte objects are in most significant byte first order. These pointer objects refer to transporter command blocks, described in Table G-1.

Field	Size	Explanation
OPERATION COMMAND CODE	8 bits	sends a message.
RESULT BLOCK POINTER	24 bits	gives the address of a data structure for the transporter to update with completion information.
SOCKET CODE	8 bits	defines which of the 4 virtual communication channels to use for this operation.
DATA BUFFER POINTER	24 bits	gives the address of a message buffer for this operation.
DATA LENGTH FIELD	16 bits	gives the length of the message to be transmitted or maximum message length accepted, if this is a receive operation. The maximum length allowed for a single message packet is 2048.
CONTROL FIELD	8 bits	gives the length of an independent auxiliary message that can be sent to a special CONTROL buffer in the destination host at an address different from

LENGTH		that of the destination message buffer. In the case of a receive command, this field specifies the largest such CONTROL message acceptable.
DESTINATION HOST	8 bits	specifies network address of the target host. Legal network addresses are 0-63, or 255 for broadcast messages. A host's address is set by switches connected to the transporter hardware.

Table G-1. Transporter Command Block

Not all fields are used by all commands, but the syntax of the command block is usually consistent, except in the case of special diagnostic commands.

The result pointer in the command block must contain the address of a large enough data structure in host memory to accept the completion information that the specified command produces. Note that the result block is associated with the operation the command block describes. If more than one operation is posted to the transporter hardware, each must have its own result block available. Table G-2 describes a typical result block.

Field	Size	Explanation
OPERATION STATUS CODE	8 bits	set to 254 by the transporter processor once it has read and accepted the command block. This field is later set by the transporter to a result code when it has completed the requested operation.
SOURCE HOST NUMBER	8 bits	gives the network address of the node from which this message packet came.
ACTUAL DATA LENGTH	16 bits	gives the actual length of the message in the receive buffer.
CONTROL MESSAGE BUFFER	0-255 bytes	a buffer large enough to accept any CONTROL message transmitted with the main message packet. The command block that points to this result block must allow such messages.

Table G-2. Receive Result Block

Up to four simultaneous receive operations can be in progress at any one time, waiting for messages for the four logical sockets in the host. Only one message can be posted for transmission at any one time, but this can be done even while four receive operations are pending. Messages from one node are only acceptable to another node if it has a receive command outstanding specifying the socket to which the message is directed.

In use, the host processor must build a command block, then post it to the transporter hardware by outputting one byte at a time of its 24-bit address to the transporter via an output port. The transporter uses an input ready status bit to synchronize this transfer. Command pointers can be transfers done at any time except while the transporter is processing a command block to transmit a message. That operation ties up the transporter until the message has been delivered, or the transporter has given up trying. Network latency is low, so the transporter is unavailable only briefly.

Once the transporter has read and accepted a command, it sets the operation status code in the result block to

254. It is advisable for the host to preset this byte to 255 before sending the transporter the pointer, so that the transporter can confirm that the command was accepted by checking for the change.

The host then polls all active result blocks, waiting for any operation status code to change to a value other than 0FEh. This change means the transporter has completed the operation associated with that result block, and data and result information are available. To simplify interpretation of results, all error codes are between 80h and 0FEh, and all success codes are less than 80h. Send and receive calls that succeed give the number of retries as a completion code, but this code is always less than 7Fh.

OMNINET transporter interfaces usually support generation of a host interrupt whenever the transporter writes to a result block. This relieves the host of having to poll result blocks for completion. To simplify OMNINET evaluation, the engineering transporter is not usually configured to use interrupts. The sample programs demonstrate the use of the transporter both without interrupts and with external interrupt hardware. Servers usually need interrupt hardware or an XIOS polling routine to achieve a usable throughput, but the sample drivers can be made to run without either if high throughput is not a goal.

The coprocessor interface structure the transporter uses is close to the ideal model of a perfect transport layer. The transporter hardware deals with all retries, message acknowledgments, packet sequencing checking, and error detection totally transparently to the host it serves. The data-structure based message interface between the host and transport layer is useful even in implementing non-OMNINET interrupt-driven transport layers for CP/NET.

G.2 Implementation Structure

In the sample implementation, very few OMNINET features were needed. All CP/NET traffic is on one logical channel (SOCKET 2), leaving the others free for such non-CP/NET uses as providing bootstrap channels between diskless devices and optional processes to load them, providing non-CP/NET peripheral sharing routines or even supporting a second network operating system in concurrent use.

Because CP/NET processes its own control fields (message headers), the control message options are not used and are set to zero. In the evaluation transporter, the most significant byte of the memory address is not used and is always set to zero. Other hardware implementations can use this byte for segment control to allow the message buffers to be banked out, or for a 16-bit processor.

The network node ID of an OMNINET host is set by six switches on its transporter hardware. In this implementation, the NODE number is the CP/NET network ID. Set the ID of the SERVER to 00. A requester can have any other unique OMNINET ID code except 0FF hex. This ID code freedom is achieved by a routine in the NETWRKIF module that binds requester ID codes dynamically to processes in the SERVER.RSP module by tracking login and logoff messages. Hence, up to 63 requesters can be supported, as long as no more than NSLAVES are logged in at any one time. Because the transporter handles all low-level communication concerns, the NETWRKIF module is relatively compact; and 16 requesters are easily supported in most systems.

To simplify coding the interface modules, data structure constructor macros eliminate the need for typing all the definitions again and again for each requester. This technique requires that the indices into the resulting arrays of data structures be computed at run-time, but this is easy to do and, where possible, is part of initialization.

G.3 The SNIOS Implementation

The intelligent nature of the OMNINET interface makes coding the SNIOS a simple exercise. Allocate a set of prefabricated transporter command blocks and associated result blocks. Even though the requester never has more than one operation pending at a time, it is simpler to use separate command blocks for each needed operation type than to recycle the same command block.

Unfortunately, relocating 8080 assemblers like RMAC do not easily deal with relocation of multibyte pointers that are not in Intel® standard memory order. It is simplest to set the result block pointers at initialization; that approach is used here.

After setting up these pointers, the NTWRKINIT routine posts a prebuilt transporter command block called INITTCB to the transporter via the routine called OMNI$STROBE. If the transporter does not accept the pointer, initialization aborts and an error returns to the NDOS. If the transporter accepts the pointer, NTWRKINIT calls OMNI$WFDONE to poll the result block associated with INITTCB until the transporter reports a completion. If the initialization operation succeeds, the node number presently set into the transporter's switches is found as a result code. If initialization fails, a value > 80h corresponding to an error code is found and returned to NTWRKINIT, and NTWRKINIT aborts and returns an error code to the NDOS. Otherwise, the node number returned is installed in configtbl and the default message buffer's SID field, the requester ID and a banner print on the console, and a success code is returned to the NDOS.

The NTWRKERROR entry is functionally identical to NTWRKINIT except that it does not print a banner or requester ID code.

The NTWRKSTS, CNFGTBLADR, and NTWRKWBOOT routines are identical in function and operation to those used with other transport layers.

When the NDOS calls the SENDMSG routine, the BC register pair contains a pointer to the message to be sent on the network. This routine translates the CP/NET header information of that message into a form consistent with OMNINET and then puts it into a prefabricated transporter command block called TXTCB. The CP/NET DID is used as the target node physical address on the network. The address of the whole message, including the CP/NET header, is placed in the buffer field of TXTCB after the pointer is rearranged into MSB, LSB sequence. The CP/NET SIZ field is adjusted to give the total message length, including the CP/NET header, and is placed in the appropriate field of the TXTCB.

The OMNINET interface primitives OMNI$STROBE and OMNI$WFDONE again post the command to the transporter and, if successful, await completion of the transmission operation. The completion code is transformed into a flag the NDOS expects. Because a very busy server might not have a buffer posted when the requester sends the message, even though 'the transporter does multiple retries by itself, a retry loop tries to send the message again, if necessary. In practice, retries are rare, but the retry loop is useful when debugging a server.

Like SENDMSG, the RECEIVEMSG routine is primarily an exercise in the translation of parameters and their transmission to the transporter. The operation of RECEIVEMSG is easily understood by reading its code, with one exception; if a receive is posted, and no message ever comes in, the transporter waits forever for a message. To simplify debugging and recovery from network errors, the OMNI$WFDONE routine times out after about 20 seconds (on a 2 mhz processor) and returns an error flag to its caller. Most servers ordinarily respond in this time, so the RECEIVEMSG routine issues a cancel receive command to the transporter via a prefabricated command block called UNRXTCB. RECEIVEMSG then returns to the NDOS with an error code.

If the receive call is not cancelled, an unsolicited or late message might be written into host memory at the requested address long after the host is using that memory for something else. Most autonomous transport layers support this kind of cancellation.

The implementation here is less than 280h bytes long, including the default 138-byte message buffer. If space is tight, the message printing and banner routines can be placed in the default buffer, a single transporter command block and result block can be recycled for all commands, and concessions to modularity can be made to yield an even smaller SNIOS.

G.4 The NETWRKIF Implementation Model

This sample OMNINET NETWRKIF uses a slightly different intermodule communication model from the one usually used to implement a serial asynchronous star network. Instead of using one process per server process to implement the network input and output, a single input process and a single output process route all messages. This type of structure is far more efficient for any party-line type of network interface hardware because fewer dispatches occur per transaction. Those transactions that do occur take less time and far less code is required to implement the NETWRKIF. In addition, the structure is easier to understand and debug, and all traffic converges through one piece of code, allowing you to implement message routing extensions to your network.

This model is easily understood by studying the general function of the network receiver and transmitter process separately.

The network receiver process in this version is named SERVERX. It is responsible for collecting each incoming message as it arrives, identifying the server process it is for, and writing a pointer to the message into that process's input queue. In addition, SERVERX functions as a surrogate server process to advise requesters that are not logged in that they have no server process to use.

SERVERX uses run-time binding of requester ID codes to server processes. SERVERX does this by keeping a table of the input queue addresses of all the server processes it supports and the ID code of the requester currently logged in to each process. SERVERX examines each incoming messages SID field and searches the table to find out whether SID is presently associated with a server process. If not, an error reply message is constructed in the same buffer that the message arrived in, and SERVERX writes this message directly to the network output process for transmission back to the requester.

For this process to function properly, SERVERX must track all login and logoff messages that pass through it. Every time a login message is received, SERVERX checks its mapping table to find out whether that requester is currently associated with a server process. If it is, no action is taken. If not, SERVERX tries to find an idle server entry in the table. Idle entries are shown in this table as in use by requester 255. If a free server entry is located, SERVERX enters the requester's ID into it, and then sends the login message to that server process's input queue. If none are available, an error reply message is constructed by SERVERX and sent back to the requester.

Logoff messages are handled by finding that requester's server entry, marking it as empty (255) , and then routing the logoff message to the server's input queue. If that requester was never logged in in the first place, SERVERX sends it an error, as previously explained.

Because there is no way to know which server process an incoming message will be for at the time a buffer is posted to the transporter for a receive call, buffers are not permanently assigned to particular server processes. Instead, a list of empty buffers is kept in an MP/M II queue, and SERVERX obtains the buffers from the queue as needed and available for posting to the transporter.

The OMNINET primitives are similar to those used by the SNIOS, except that an MX queue ensures that the transporter is not in use by another process when SERVERX wants to post a command block pointer to it.

As the arrival time of the next message is unknown, SERVERX must be suspended while it waits for the next message to arrive. This can be done by an XDOS flag wait in the WF$RXDONE OMNINET primitive or by delay-based polling. If your XIOS can be easily modified, another alternative is to add an XIOS polling routine. Using the delay call to suspend the process drastically reduces network throughput because only 60 incoming messages can arrive per second.

The SERVETX process is extremely simple. It reads messages from a single input queue and posts them, using mutual exclusion, to the transporter. Because messages are quickly disposed of by the network, there is no point in suspending SERVETX. It uses a different completion routine than SERVERX, which merely waits until a completion code is received from the transporter, and then returns to its caller. To simplify debugging, a timeout is included to prevent a hardware or software problem from locking up the system.

Once SERVETX has finished sending the message, it returns the buffer that it was in to the free buffer management queue, making it available for SERVERX. SERVETX then goes back to read its input queue to wait for another message to process.

Theoretically, such a system can function with fewer buffers than server processes. But in practice, it is best to have at least one more buffer than the number of server processes in the pool to deal with messages such as failed login attempts that never get routed to a server.

The rest of the code in each process simply initializes data structures, creates queues, initializes hardware, and performs other routine tasks.

Note that the distribution version of CP/NET 1.2 does not work with this SERVETX process without a minor patch. SERVER.RSP must be patched to create output UQCBs with the same name for all server processes instead of making each queue name unique. Once this is done, all processes in SERVER.RSP direct their output to a single SERVETX process. Instructions for installing this patch are included in *CP/NET V1.2 Application Note 02*.

G.5 Possible Improvements to NETWRKIF

This interface is by no means ideal. Little error recovery is done for registers that fail to log off. A watchdog timing process can be easily added to correct this problem. This process is not shown here, to simplify understanding of the OMNINET interface. But such a process is only needed in systems with more physical requesters than server processes to prevent their being locked up by departed users.

One possible improvement is to further reduce the number of dispatches per CP/NET transaction by using direct code to manage the buffer list and using the transporter mutual exclusion function instead of the MP/M II queue facility. The M/PM II queue facility is powerful and easy to use, but avoid using it in situations where dispatch overhead exceeds the time for which a process is likely to require suspension unless the suspension is unavoidable for process synchronization reasons.

Another worthwhile improvement is to modify the NETWRKIF to minimize the period during which the server cannot respond to incoming messages, by seeing that the next buffer is more quickly posted for the next received message after a receive completion occurs. The present version does not do this until the incoming message has been processed by SERVERX. This causes unneeded network traffic because messages sent by requesters during this time are futile.

High-performance servers can make good use of two physical sets of transporter hardware, with different node addresses, on the same loop. Using two transporters can totally bypass the need to use MX techniques because one transporter can be reserved solely for transmitting messages.

Interesting networks can be easily constructed by having more than one OMNINET loop, each with its own transporter. The SERVERX process associated with each loop can filter messages not intended for local SLVSPs to a second, third, or fourth SERVETX process associated with higher level loops. Such filtering bridges can be used to build hierarchical CP/NET systems of any degree of complexity.

Other processes can concurrently send and receive messages totally unrelated to the CP/NET context using the same transporter as long as they honor the MXomni mutual exclusion queues and do not use the same socket for their communication as CP/NET. These processes can implement a variety of supervisory and auxiliary functions, or they can implement additional concurrent virtual circuits that cooperating requesters can use for point-to-point traffic. Such point-to-point virtual circuits can be coordinated by CP/NET mail functions.

```
CP/M RMAC ASSEM 1.1     #001    SAMPLE SLAVE NETWORK I/O SYSTEM FOR CORVUS OMNINET 20 OCT 82

                          title 'Sample Slave Network I/O System for CORVUS OMNINET 20 Oct 82'
                                page    54

                        ;________________________________________________________________________
                        ;________________________________________________________________________
                        ;                                                                       ;
                        ;       SAMPLE SLAVE NETWORK IO SYSTEM FOR CP/NET 1.2                   ;
                        ;       VERSION FOR CORVUS OMNINET "ENGINEERING" TRANSPORTER            ;
                        ;               (Requires RMAC for assembly)                            ;
                        ;                                                                       ;
                        ;       COPYRIGHT (C) 1982 by   VANO ASSOCIATES, INC.                   ;
                        ;                               P.O. BOX 12730                          ;
                        ;                               New Brighton, MN  55112                 ;
                        ;                               U.S.A.                                  ;
                        ;                               (612) 631-1245                          ;
                        ;                               ALL RIGHTS RESERVED                     ;
                        ;                                                                       ;
                        ;       ANY USE OF THIS CODE without the imbedded copyright notice and  ;
                        ;       banner is hereby strictly prohibited.                           ;
                        ;                                                                       ;
                        ;       Permission is hereby granted to Digital Research Inc. to use    ;
                        ;       this source file for educational and illustrative purposes in   ;
                        ;       conjunction with CP/Net 80 documentation.  Any other use of     ;
                        ;       this code without the EXPRESS WRITTEN PERMISSION of VANO        ;
                        ;       ASSOCIATES INC. is hereby strictly prohibited.                  ;
                        ;                                                                       ;
                        ;       This file is provided courtesy of:                              ;
                        ;                                                                       ;
                        ;                       R2E (Realisations Etude Electroniques)          ;
                        ;                       Z.A.I. de Courtaboeuf                           ;
                        ;                       BP 73   91942 Les Ulis                          ;
                        ;                       FRANCE                                          ;
                        ;                                                                       ;
                        ;       who sponsored the development of one of its ancestors.          ;
                        ;________________________________________________________________________
                        ;________________________________________________________________________

                        ;       *****   CONSTANT DECLARATIONS   *****

        0000 =          FALSE   equ     0
        FFFF =          TRUE    equ     not FALSE

                        ;       configuration and option constants
        0064 =          TXTRIES equ     100             ;Transmit message retries
        008A =          BUFFSIZE equ    138             ;max default buffer size
        0200 =          MAXMSG  equ     512             ;largest message accepted by receiver
        0080 =          SKT0    equ     80h             ;legal omninet socket tokens
        0090 =            SKT1  equ     90h
        00A0 =            SKT2  equ     0a0h
        00B0 =            SKT3  equ     0b0h
        00A0 =          SOCKET  equ     SKT2            ;this SNIOS uses only channel 2

                        ;       OMININET Constants
                        ;  Completion/return codes

CP/M RMAC ASSEM 1.1     #002    SAMPLE SLAVE NETWORK I/O SYSTEM FOR CORVUS OMNINET 20 OCT 82

        0000 =          NOERR   equ     0               ;done (no errors or retries)
        00C0 =          ETXOK   equ     0c0h            ;echo succeeded with no retries (not used here)
        0080 =          ETXFAIL equ     80h             ;Transmit failed
        0081 =          E2LONG  equ     81h             ;wouldn't fit in destination socket
        0082 =          ENOSKT  equ     82h             ;destination socket not set up
        0083 =          EBDCTL  equ     83h             ;bad control field length
        0084 =          EBDSKT  equ     84h             ;illegal socket number
        0085 =          EBDDES  equ     85h             ;invalid destination node number/socket in use
        0086 =          EBDNODE equ     86h             ;bad node number in command (not 0-7fh or ffh)
        00FE =          ECMDOK  equ     0feh            ;command has been read by transporter
```

```
                ; legal command tokens
0040 =          SENDF   equ     40h             ;send message
00F0 =          RCVF    equ     0f0h            ;set up receive socket
0010 =          ENDRCVF equ     10h             ;stop receive
0020 =          INITF   equ     20h             ;initialize transporter
                ; Transporter control ports
00F8 =          NETBASE equ     0f8h            ;base address of transporter IO interface
00F9 =            TSTAT equ     Netbase+1       ;ready status port
0010 =            TCRDY equ     10h             ;status mask for ready bit
00F8 =            TDATA equ     Netbase         ;command block pointer port

                ;       Network Status Byte Constants
                ;
0010 =          ACTIVE          equ     10h     ;slave logged in on network
0002 =          RCVERR          equ     2h      ;error in received message
0001 =          SENDERR         equ     1h      ;unable to send message

                ;       CP/M BDOS function constants
0005 =          BDOS    equ     5               ;absolute BDOS entry
0009 =          PRINTF  equ     9               ;print message function
0002 =          CONOUTF equ     2               ;output char in E to console

                ;       General Constants
000A =          LF      equ     0ah             ;Line Feed
000D =          CR      equ     0dh             ;Carriage Return

                ;       *****   GENERATED CODE AND DATA BEGIN HERE      *****

                ;       Public Jump vector for SNIOS entry points
0000 C3F400             jmp     ntwrkinit       ;network initialization
0003 C34801             jmp     ntwrksts        ;network status
0006 C35201             jmp     cnfgtbladr      ;return config table addr
0009 C36701             jmp     sendmsg         ;send message on network
000C C3A601             jmp     receivemsg      ;receive message from network
000F C33801             jmp     ntwrkerror      ;network error
0012 C35601             jmp     ntwrkwboot      ;network warm boot

                ;       Public Slave Configuration Table
                configtbl:
                Network$status:
0015 00                 db      0                               ;network status byte
0016 00         slvid1: db      0                               ;slave ID (from switches)
0017 0000000000         db      0,0,    0,0,    0,0,    0,0     ;Disk map table for units A:-P:
001F 0000000000           db    0,0,    0,0,    0,0,    0,0
```



```
0027 0000000000           db    0,0,    0,0,    0,0,    0,0
002F 0000000000           db    0,0,    0,0,    0,0,    0,0
0037 0000               db      0,0                             ;console device
0039 0000               db      0,0                             ;list device
003B 00                 db      0                               ;buffer index
                ;
003C 00         dflt:   db      0                               ;FMT (DEFAULT MESSAGE BUFFER)
003D 00                   db    0                               ;DID
003E 00         slvid2:   db    0                               ;SID
003F 05                   db    5                               ;FNC
0040 00                   db    0                               ;SIZ
0041                      ds    1                               ;MSG(0)  List number
0042                      ds    BUFFSIZE                        ;MSG(1) ... MSG(128)


                ;       *****   PREFABRICATED OMNINET TRANSPORTER COMMAND BLOCKS   *****

                ;       Command block for transmitting a message
                TXtcb:
00CC 40          TXtcmd:        db      SENDF           ;command field
00CD 00                         db      0               ;bits 16-24 of result block ptr
00CE 0000        TXtrslt:       db      0,0             ;result block pointer (MSB,LSB)
00D0 A0          TXtskt:        db      SOCKET          ;socket (channel)  number
00D1 00                         db      0               ;bits  16-24 of message buffer ptr
00D2 0000        TXtmsg:        db      0,0             ;message buffer pointer (MSB,LSB)
00D4 0000        TXtdlen:       db      0,0             ;data field length (MSB,LSB)
00D6 00          TXtclen:       db      0               ;control field length
00D7 00          TXtdest:       db      0               ;Destination address (transport layers)
                ;       Result vector for above command block
                TXresult:
```

```
00D8 00         TXrcode:       db      0               ;return code

               ;       Command block for setting up a receive operation
               RXtcb:
00D9 F0         RXtcmd:        db      RCVF            ;command field
00DA 00                        db      0
00DB 0000       RXtrslt:       db      0,0             ;result block pointer (MSB,LSB)
00DD A0         RXtskt:        db      SOCKET          ;socket number
00DE 00                        db      0
00DF 0000       RXtmsg:        db      0,0             ;message address (MSB,LSB)
00E1 02         RXtdlen:       db      MAXMSG/256      ;max data field length (MSB,LSB)
00E2 00                        db      MAXMSG and 255
00E3 00         RXtclen:       db      0               ;max control field length
00E4 00         RXtdest:       db      0               ;(not used in a receive operation)
               ;       Result vector for receiver
               RXresult:
00E5 00         RXrcode:       db      0               ;return code
00E6 00         RXrsrce:       db      0               ;source HOST #
00E7 0000       RXrdlen:       db      0,0             ;received message length (MSB,LSB)

               ;       Command block for receive cancel operation
               UNRXtcb:
00E9 10         UNRXtcmd:      db      ENDRCVF         ;command field
00EA 00                        db      0
```



```
00EB 0000       UNRXtrslt:     db      0,0             ;result block pointer (MSB,LSB)
00ED A0         UNRXtskt:      db      SOCKET          ;socket number
               ;       Result vector for receive cancel
               UNRXresult:
00EE 00         UNRXrcode:     db      0               ;return code

               ;       Command block for transporter initialization command
               INITtcb:
00EF 20         INITtcmd:      db      INITF           ;command field
00F0 00                        db      0
00F1 0000       INITtrslt:     db      0,0             ;result block pointer (MSB,LSB)
               ;       Result vector for initialization
               INITresult:
00F3 00         INITrcode:     db      0               ;return code (if valid,=ID code)


               ;       *****   PUBLIC CODE ENTRIES BEGIN HERE   *****

               ;       Externally accessed routine to initialize transporter
               ;               (RETURNS A=0 if succeeds, else 0ffh.)
               ntwrkinit:
00F4 CD3801            call    ntwrkerror              ;init transporter, tcbs and id code
00F7 D8                rc                              ;return error if init fails
00F8 110601            lxi     d,initmsg               ;else prinw slave ID and banner
00FB CDF001            call    print$msg
00FE 3A1600            lda     slvid1
0101 CDD601            call    prhex                   ;print slave ID
0104 AF                xra     a                       ;and return to caller with a=0
0105 C9                ret

               initmsg:
0106 0D0A534E49        db      CR,LF,'SNIOS (c)1982 Vano Associates Inc.'
012A 0D0A534C41        db      CR,LF,'SLAVE ID = $'


               ;       Externally accessed routine inits or re-inits module
               ;               (RETURNS A=0 if succeeds, else 0ffh.)
               ntwrkerror:
0138 AF                xra     a
0139 321500            sta     Network$status          ;zero network status byte
013C CDF501            call    omni$init               ;init transporter, tcbs and id code
013F D8                rc                              ;carry means error, A=0ffh
0140 321600            sta     slvid1                  ;update this slaves id in table
0143 323E00            sta     slvid2                  ;and default message
0146 AF                xra     a                       ;and return with no error
0147 C9                ret


               ;       Externally accessed routine returns Network Status Byte in A
               ;               (also clears any error bits active)
```

```
                ntwrksts:
0148 211500             lxi     h,network$status
014B 46                 mov     b,m
014C 3EFC               mvi     a,not(RCVERR or SENDERR)
```



```
014E A0                 ana     b
014F 77                 mov     m,a
0150 78                 mov     a,b
0151 C9                 ret


                ;       Externally accessed routine Returns Configuration Table Ptr in HL
                cnfgtbladr:
0152 211500             lxi     h,configtbl
0155 C9                 ret


                ;       Externally accessed routine is called each time the CCP is reloaded
                ;       from disk.  (Dummy procedure for now.)
                ntwrkwboot:
0156 115C01             lxi     d,wboot$msg             ;return via print$msg
0159 C3F001             jmp     print$msg

                wboot$msg:
015C 0D0A3C4350         db      CR,LF,'$'


                ;       Externally accessed routine sends Message BC--> on Network
                ;               (returns A=0 if succeeds, else A=0ffh.)
                ;
                ;       NOTE that although the OMNINET transporter does its own transport
                ;         layer retries, this routine does additional retries to deal with
                ;         servers that are slow in posting receive calls since transport
                ;         level retries are exhausted in a very short real-time period.
                sendmsg:
0167 61                 mov     h,c             ;move buffer pointer to Transporter ctrl block
0168 68                 mov     l,b             ;(note reversed byte order for Transporter.)
0169 22D200             shld    TXtmsg
                ;
016C 210400             lxi     h,4             ;get CP/Net message length from SIZ field
016F 09                 dad     b
0170 6E                 mov     l,m
0171 2600               mvi     h,0
0173 110600             lxi     d,6             ;add packet header lgth to get actual size
0176 19                 dad     d               ;  of packet for transport layer purposes
0177 7C                 mov     a,h             ;swap bytes to MSB, LSB order
0178 65                 mov     h,l
0179 6F                 mov     l,a
017A 22D400             shld    TXtdlen         ;store length in TCB data length field
                ;
017D 03                 inx     b               ;get DID from message
017E 0A                 ldax    b
017F 32D700             sta     TXtdest         ;put it into TCB destination address field
                ;
0182 116400             lxi     d,TXTRIES       ;use DE as retry counter
                ;
                send$again:                     ;head of message transmission retry loop
0185 D5                 push    d
0186 01CC00             lxi     b,TXtcb         ;send TCB pointer to transporter hardware
```



```
0189 CD2E02             call    omni$strobe
018C D1                 pop     d
018D DAA101             jc      snderr          ;if not accepted, goto fatal error handler
                ;
0190 01D800             lxi     b,TXresult      ;else poll result block until completion code
0193 D5                 push    d               ;is returned by hardware
0194 CD5C02             call    omni$wfdone
0197 D1                 pop     d
                ;
0198 E680               ani     80h             ;completion codes 80h-ffh are error codes
019A C8                 rz                      ;return 00 to caller if no errors
                ;
```

```
019B 1B                 dcx     d               ;else decrement retry counter
019C 7B                 mov     a,e
019D B2                 ora     d
019E C28501             jnz     send$again      ;retry transmit if any retries left
                ;
01A1 3E01       snderr: mvi     a,SENDERR       ;goto common exit code to update error flags
01A3 C3CE01             jmp     nerr            ;(part of receivemsg routine)
                ;       Externally accessed routine waits for a message directed to this node
                ;       and returns it in the buffer BC-->.  To aid debugging, a timeout of
                ;       about 20 seconds (2 Mhz processor) is implemented that will return an
                ;       error if no message is received.  That is long enough for most normal
                ;       servers to respond.
                ;
                ;       (RETURNS A=0 if good msg, =0ffh if  bad msg or timeout.)
                receivemsg:
01A6 68                 mov     l,b             ;swap buffer pointer bytes to MSB,LSB order
01A7 61                 mov     h,c
01A8 22DF00             shld    RXtmsg          ;put buffer ptr to its TCB field
                ;
01AB 01D900             lxi     b,RXtcb
01AE CD2E02             call    omni$strobe     ;post control block address to hardware
01B1 DACC01             jc      rxerr           ;fatal error if hardware won't accept it
                ;
01B4 01E500             lxi     b,RXresult
01B7 CD5C02             call    omni$wfdone     ;else wait for a completion from hardware
01BA E680               ani     80h
01BC C8                 rz                      ;return 00 to caller if no error reported
                ;  the rest is the fatal error handler for receive calls
01BD 01E900             lxi     b,UNRXtcb       ;otherwise cancel the receive call
01C0 CD2E02             call    omni$strobe     ; (using prefabricated cancel command block)
01C3 D2CC01             jnc     rxerr           ;If won't accept this command either, quit here
                ;
01C6 01EE00             lxi     b,UNRXresult    ;else wait for completion of cancel command
01C9 CD5C02             call    omni$wfdone     ;ignore result (always fatal error return)
01CC 3E02       rxerr:  mvi     a,RCVERR        ;exit via code that updates status byte
                ;       This is also used by sendmsg to update Network$status and return 0ffh
01CE 211500     nerr:   lxi     h,Network$status
01D1 B6                 ora     m
01D2 77                 mov     m,a             ;update status
```



```
01D3 3EFF               mvi     a,0ffh
01D5 C9                 ret                     ;return 0ffh to caller
                ;       *****   UTILITY ROUTINES CALLED BY ABOVE BEGIN HERE  *****
                ;       prints A in hex on console
01D6 F5         prhex:  push    psw
01D7 07                 rlc
01D8 07                 rlc
01D9 07                 rlc
01DA 07                 rlc
01DB CDDF01             call    nibl            ;print high nibble
01DE F1                 pop     psw             ;and fall through to  print low nibble
01DF E60F       nibl:   ani     0fh
01E1 C630               adi     '0'
01E3 FE3A               cpi     '9'+1
01E5 DAEA01             jc      printa
01E8 C607               adi     7
01EA 5F         printa: mov     e,a
01EB 0E02               mvi     c,CONOUTF
01ED C30500             jmp     BDOS            ;print ascii and return
                ;       print message DE--> until $ on console device
                print$msg:
01F0 0E09               mvi     c,PRINTF                ;prints $ delimited string DE-->
01F2 C30500             jmp     BDOS                    ;bdos(printf,wboot$msg)
                ;       *****   LOW LEVEL OMNINET TRANSPORTER DRIVERS BEGIN HERE   *****
```

```
                ;       Initialize transporter and return its ID code in A or 0ffh if can't.
                ;       Carry is also set if error, clear if no error.
                omni$init:                              ;initialize pointers in our control blocks
01F5 11D800             lxi     d,TXresult      ;NOTE: this is done at run time to avoid
01F8 63                 mov     h,e             ;  relocation problems caused by the need to
01F9 6A                 mov     l,d             ;  have pointers for CORVUS transporter use
01FA 22CE00             shld    TXtrslt         ;  in MSB, LSB form instead of 8080 format.
                ;
01FD 11E500             lxi     d,RXresult
0200 63                 mov     h,e
0201 6A                 mov     l,d
0202 22DB00             shld    RXtrslt
                ;
0205 11EE00             lxi     d,UNRXresult
0208 63                 mov     h,e
0209 6A                 mov     l,d
020A 22EB00             shld    UNRXtrslt
                ;
020D 11F300             lxi     d,INITresult
0210 63                 mov     h,e
0211 6A                 mov     l,d
```



```
0212 22F100             shld    INITtrslt
                ;
0215 01EF00             lxi     b,INITtcb       ;send init command block pointer to transporter
0218 CD2E02             call    omnistrobe      ;to reset it and get its ID code
021B 9F                 sbb     a               ;in case of error, preset return code 0 or ff
021C D8                 rc                      ;fatal error if hardware won't accept pointer
                ;
021D 01F300             lxi     b,INITresult    ;else wait for result of operation
0220 CD5C02             call    omni$wfdone     ;wait for done
0223 321600             sta     slvid1          ;result code should be ID code so put in table
0226 323E00             sta     slvid2          ;and in default message SID
                ;
0229 07                 rlc                     ;set CY=bit 7 of return code
022A 1F                 rar                     ;so CY=1 if error
022B D0                 rnc                     ;return with ID code if no error
022C 9F                 sbb     a               ;else set carry=1 and A=0ffh and return
022D C9                 ret


                ;       Sends the 16 bit POINTER in BC to the transporter hardware as
                ;       a 24 bit pointer (MSB first).  Returns CY set if hardware will
                ;       not accept any byte in a reasonable time else CY clear.
                omni$strobe:
022E 210200             lxi     h,2             ;Find address of rslt block from TCB BC-->
0231 09                 dad     b               ;pre-set result code in block to ff (busy)
0232 7E                 mov     a,m
0233 23                 inx     h
0234 6E                 mov     l,m
0235 67                 mov     h,a
0236 36FF               mvi     m,0ffh
                ;
0238 AF                 xra     a               ;MSB is always 0
0239 CD4302             call    omni$st         ;send bits 23-16 of pointer to hardware
023C D8                 rc                      ;(abort if timeout)
                ;
023D 78                 mov     a,b             ;send bits 15-8 of pointer to hardware
023E CD4302             call    omni$st
0241 D8                 rc                      ;(abort if timeout)
                ;
0242 79                 mov     a,c             ;send bits 7-0 of pointer to hardware
                                                ; (fall into omni$st)

                ;       called by omni$strobe to send one byte from A to transporter hardware
                ;       returns CY set if hardware doesn't come ready in a reasonable time.
                omni$st:
0243 F5                 push    psw             ;save data for now
0244 1150C3             lxi     d,50000         ;set timeout
                omni$st0:
0247 DBF9               in      TSTAT           ;read status port and check busy bit
0249 E610               ani     TCRDY
024B CA5302             jz      omni$st1        ;if busy, go increment and test timeout
                ;
```

```
024E F1                 pop     psw             ;else output the byte
024F D3F8               out     TDATA           ;to the transporter TCB pointer input register
```



```
0251 B7                 ora     a
0252 C9                 ret                     ;and return with no error shown (CY=0)
                ;
                omni$st1:                       ;else
0253 1B                 dcx     d
0254 7A                 mov     a,d
0255 B3                 ora     e
0256 C24702             jnz     omni$st0        ;loop back if not timed out yet
                ;
0259 F1                 pop     psw             ;else
025A 37                 stc
025B C9                 ret                     ;return error flag (CY=1)


                ;       waits till timeout (about 20 secs) for result block BC--> to show done
                ;       returns A=returned status code.  If timeout occurs, the returned
                ;       status will still be 0FEH or 0FFH.
                omni$wfdone:
025C 11FFFF             lxi     d,0ffffh        ;setup timeout counters
025F 2E14               mvi     l,20
                ;
                omni$wfdone1:
0261 0A                 ldax    b               ;is the result code still > 0f0h?
0262 FEF0               cpi     0f0h
0264 D8                 rc                      ;no, return to caller
                ;
0265 1B                 dcx     d               ;else decrement timeout
0266 7B                 mov     a,e
0267 B2                 ora     d
0268 C26102             jnz     omni$wfdone1    ;timeout yet?
026B 2D                 dcr     l
026C C26102             jnz     omni$wfdone1    ;no, go back and check again
                ;
026F 0A                 ldax    b               ;yes, timeout
0270 C9                 ret                     ;return with completion code in A


0271                    end
```



```
ACTIVE           0010    78#
BDOS             0005    83#  347   353
BUFFSIZE         008A    45#  121
CNFGTBLADR       0152    96   224#
CONFIGTBL        0015   103#  225
CONOUTF          0002    85#  346
CR               000D    89#  194   195   236
DFLT             003C   115#
E2LONG           0081    58#
EBDCTL           0083    60#
EBDDES           0085    62#
EBDNODE          0086    63#
EBDSKT           0084    61#
ECMDOK           00FE    64#
ENDRCVF          0010    68#  161
ENOSKT           0082    59#
ETXFAIL          0080    57#
ETXOK            00C0    56#
FALSE            0000    40#   41
INITF            0020    69#  171
INITMSG          0106   186   193#
INITRCODE        00F3   176#
INITRESULT       00F3   175#  376   386
INITTCB          00EF   170#  381
INITTCMD         00EF   171#
INITTRSLT        00F1   173#  379
LF               000A    88#  194   195   236
MAXMSG           0200    46#  149   150
NERR             01CE   289   322#
NETBASE          00F8    71#   72    74
```

```
NETWORKSTATUS    0015   104#  202   214   322
NIBL             01DF   337   340#
NOERR            0000    55#
NTWRKERROR       0138    99   184   200#
NTWRKINIT        00F4    94   183#
NTWRKSTS         0148    95   213#
NTWRKWBOOT       0156   100   231#
OMNIINIT         01F5   203   360#
OMNIST           0243   411   415   423#
OMNIST0          0247   426#  440
OMNIST1          0253   429   436#
OMNISTROBE       022E   271   305   314   382   401#
OMNIWFDONE       025C   277   309   318   387   450#
OMNIWFDONE1      0261   454#  462   464
PRHEX            01D6   189   332#
PRINTA           01EA   343   345#
PRINTF           0009    84#  352
PRINTMSG         01F0   187   233   351#
RCVERR           0002    79#  216   319
RCVF             00F0    67#  143
RECEIVEMSG       01A6    98   299#
RXERR            01CC   306   315   319#
RXRCODE          00E5   155#
RXRDLEN          00E7   157#
```

```
CP/M RMAC ASSEM 1.1     #011    SAMPLE SLAVE NETWORK I/O SYSTEM FOR CORVUS OMNINET 20 OCT 82

RXRESULT         00E5   154#  308   366
RXRSRCE          00E6   156#
RXTCB            00D9   142#  304
RXTCLEN          00E3   151#
RXTCMD           00D9   143#
RXTDEST          00E4   152#
RXTDLEN          00E1   149#
RXTMSG           00DF   148#  302
RXTRSLT          00DB   145#  369
RXTSKT           00DD   146#
SENDAGAIN        0185   268#  286
SENDERR          0001    80#  216   288
SENDF            0040    66#  128
SENDMSG          0167    97   246#
SKT0             0080    47#
SKT1             0090    48#
SKT2             00A0    49#   51
SKT3             00B0    50#
SLVID1           0016   106#  188   205   388
SLVID2           003E   117#  206   389
SNDERR           01A1   273   288#
SOCKET           00A0    51#  131   146   164
TCRDY            0010    73#  428
TDATA            00F8    74#  432
TRUE             FFFF    41#
TSTAT            00F9    72#  427
TXRCODE          00D8   139#
TXRESULT         00D8   138#  275   361
TXTCB            00CC   127#  270
TXTCLEN          00D6   135#
TXTCMD           00CC   128#
TXTDEST          00D7   136#  264
TXTDLEN          00D4   134#  260
TXTMSG           00D2   133#  249
TXTRIES          0064    44#  266
TXTRSLT          00CE   130#  364
TXTSKT           00D0   131#
UNRXRCODE        00EE   167#
UNRXRESULT       00EE   166#  317   371
UNRXTCB          00E9   160#  313
UNRXTCMD         00E9   161#
UNRXTRSLT        00EB   163#  374
UNRXTSKT         00ED   164#
WBOOTMSG         015C   232   235#
```

Listing G-1. Sample Slave Network I/O System for Corvus OMNINET

```
CP/M RMAC ASSEM 1.1     #001    SAMPLE SERVER NETWORK I/F FOR CORVUS OMNINET 20-OCT-82
```

```
                 title 'Sample Server Network I/F for CORVUS OMNINET 20-Oct-82'
                       page 54

               ;________________________________________________________________________
               ;________________________________________________________________________
               ;                                                                       ;
               ;       SAMPLE MASTER NETWORK IO SYSTEM FOR CP/NET 1.2                  ;
               ;       VERSION FOR CORVUS OMNINET "ENGINEERING" TRANSPORTER            ;
               ;               (Requires RMAC for assembly)                            ;
               ;                                                                       ;
               ;       COPYRIGHT (C) 1982 by   VANO ASSOCIATES, INC.                   ;
               ;                               P.O. BOX 12730                          ;
               ;                               New Brighton, MN  55112                 ;
               ;                               U.S.A.                                  ;
               ;                               (612) 631-1245                          ;
               ;                               ALL RIGHTS RESERVED                     ;
               ;                                                                       ;
               ;       ANY USE OF THIS CODE without the imbedded copyright notice      ;
               ;       is hereby strictly prohibited.                                  ;
               ;                                                                       ;
               ;       Permission is hereby granted to Digital Research Inc. to use    ;
               ;       this source file for educational and illustrative purposes in   ;
               ;       conjunction with CP/Net 80 documentation.  Any other use of     ;
               ;       this code without the EXPRESS WRITTEN PERMISSION of VANO        ;
               ;       ASSOCIATES INC. is hereby strictly prohibited.                  ;
               ;                                                                       ;
               ;       This file is provided courtesy of:                              ;
               ;                                                                       ;
               ;               R2E (Realisations Etude Electroniques)                  ;
               ;               Z.A.I. de Courtaboeuf                                   ;
               ;               BP 73   91942 Les Ulis                                  ;
               ;               FRANCE                                                  ;
               ;                                                                       ;
               ;       who sponsored the development of one of its ancestors.          ;
               ;                                                                       ;
               ;       Note that this version requires that the CP/NET SLAVESP         ;
               ;       process be properly patched to send all output traffic          ;
               ;       to output queue 0.  For the current (1.2) beta release, the     ;
               ;       following patch is enough:                                      ;
               ;                                                                       ;
               ;       Make this change in unrelocated server.rsp module.              ;
               ;               -a543                                                   ;
               ;               0543  mvi a,30                                          ;
               ;               0545  jmp 34f                                           ;
               ;       Then resave the module and its bit map.                         ;
               ;                                                                       ;
               ;________________________________________________________________________
               ;________________________________________________________________________

FFFF =         YES     equ     0ffffh
0000 =         NO      equ     not YES

               ;       assembly mode switches
0000 =         DEBUG           equ     NO              ;assemble for debugging with rdt
```



```
FFFF =         RSP             equ     YES             ;assemble as a resident process
0000 =         INTERRUPT       equ     NO              ;transporter can interrupt (advisable)

               ;       Logical Configuration constants
0002 =         NSLAVES         equ     2               ;maximum number of slaves supported
0096 =         SRVR$STK$SIZ    equ     150             ;stack size  needed by SLVSPs
0034 =         SRVR$PD$SIZ     equ     52              ;PD size for SLVSPs
0118 =         BUFFSIZE        equ     280             ;maximum message buffer size
0003 =         NMSG$BUFFS      equ     1+NSLAVES       ;number of message buffers allocated
0040 =         RX$PRIORITY     equ     64              ;receive process priority
003F =         TX$PRIORITY     equ     63              ;usually higher than rx

               ;       Physical configuration constants (FOR OUR INSTALLATION)
00F8 =         OMNI$BASE       equ     0F8h            ;transporter base address
00A0 =         OMNI$SOCKET     equ     0a0h            ;omninet transporter socket code (2)
0008 =         OMNI$FLAG       equ     8               ;XDOS flag for int. driven transporter
0007 =         RST$NUM         equ     7               ;interrupt level if interrupt driven
0038 =           INT$VCTR      equ     RST$NUM * 8
```

```
                  ;       transporter IO PORT constants for CORVUS "ENGINEERING" transporter
00F8 =            OMNI$DATA       equ     OMNI$BASE       ;TCB pointer data port
00F9 =            OMNI$STAT       equ     OMNI$BASE + 1   ;status port
0010 =            OMNI$RDY        equ     10h             ;ready bit (=1) in OMNI$STAT
                  ;  the rest are not part of standard CORVUS "ENGINEERING" transporter
00FA =            OMNI$ACK        equ     OMNI$BASE + 2   ;int ack port (any data write)
00FB =            OMNI$MASK       equ     OMNI$BASE + 3   ;int mask port (b0, 1= enbl)
0001 =            OMNI$PENDING    equ     1               ;int pending (=1) in "   "
0001 =            OMNI$ENABLE     equ     1               ;int enable mask command
0000 =            OMNI$DISABLE    equ     0               ;int disable mask command

                  ;       BDOS and XDOS Equates
0009 =            PRINTF          equ     9               ;message to console
0084 =            FLAGWAITF       equ     132             ;flag wait
0085 =            FLAGSETF        equ     133             ;flag set
0086 =            MAKEQ           equ     134             ;make queue
0089 =            READQ           equ     137             ;read queue
008B =            WRITEQ          equ     139             ;write queue
008D =            DELAY           equ     141             ;delay
008E =            DSPTCH          equ     142             ;dispatch
0090 =            CREATEP         equ     144             ;create process
0091 =            SET$PRIORITY    equ     145             ;set caller's priority
0093 =            DETACH          equ     147             ;detach console
009A =            SYDATAD         equ     154             ;get system data page address

                  ;       MISC useful constants
000D =            CR              equ     0dh             ;carriage return
000A =            LF              equ     0ah             ;line feed


                  codeseg:
                    if not RSP
                  ;       .PRL Initialization entry point for whole module
                          lxi     sp,ServerxSTKTOP        ;switch to rx process stack
                          mvi     c,SET$PRIORITY
```



```
                          mvi     e,RX$PRIORITY
                          call    bdos
                      if not DEBUG
                            mvi   c,DETACH
                            call  bdos                    ;detach console
                      endif ; DEBUG
                          ret

                  bdosadr:
                          dw      codeseg - 100h + 5      ;bdos entry pointer
                    else ; not RSP
                  ;       in an rsp, this is filled in by GENSYS and the rx process is created
                  ;       automatically
                  bdosadr:
0000 0000                 dw      0000h
                    endif ; not RSP

                  page
```



```

                  ;_________________________________________________________________________
                  ;_________________________________________________________________________
                  ;                                                                        ;
                  ;       This is the network receiver server process module              ;
                  ;                                                                        ;
                  ;       The receive server obtains a buffer from FreeBuff and gives it  ;
                  ;       to the transporter hardware for receive use.  It then waits     ;
                  ;       for a message completion by calling the wf$rx$done routine      ;
                  ;       Once a return from that routine occurs, the receiver server     ;
                  ;       checks the slave number and sends a pointer to that message     ;
                  ;       buffer to the SLVSP support process corresponding to that       ;
                  ;       slave's server. Once the message pointer has been passed, the   ;
                  ;       process loops back for the next message and continues in this  ;
                  ;       fashion forever.                                                ;
                  ;                                                                        ;
                  ;       At present, receive errors are considered to be the Slave's     ;
```

```
                ;       problem since normal error recovery is allegedly handled by the ;
                ;       transporter firmware.  Only error free messages are passed on,  ;
                ;       the rest are ignored unless the error is the absence of a free  ;
                ;       support process in which case a "NOT LOGGED IN" error is sent   ;
                ;       by the receiver process to the offending slave.                 ;
                ;                                                                       ;
                ;       In order to prevent clobbering the transporter when it is busy  ;
                ;       transmitting, the receiver must be synchronized with the        ;
                ;       transmit server.  In this implementation, this is handled by    ;
                ;       an MX Queue.                                                    ;
                ;                                                                       ;
                ;_______________________________________________________________________
                ;_______________________________________________________________________

                ;       receiver server process descriptor (position dependent if RSP)
                ServerxPD:
0002 0000               dw      0                       ;link
0004 0040               db      0,RX$PRIORITY           ;status,priority
0006 6400               dw      $ + 94                  ;stack pointer
0008 5365727665         db      'ServeRX '              ;name
0010 00FF               db      0,0ffh                  ;console, memseg
0012                    ds      82                      ;reserved for MP/M use and stack
                ServerxSTKTOP:
0064 9800               dw      InitRX                  ;startup PC for process

                ;       User queue control block array used by this module for message queues.
                ;        Each element is 3 words long and is one UQCB followed by its message.
0006 =          UQCBLEN equ     6                       ;constant used to index array
0004 =          XQCBMSG equ     4                       ;subindex for  message word

                INUQCB:                                 ;array name
0000 #          ??xx    set     0
                        rept    NSLAVES
                          dw    (inqcb$array + ??xx)    ;;Q pointer, msg addr, message word
                          dw    $+2
                          dw    0
                ??xx      set   ??xx + INQCB$SIZE
```



```
                        endm
0066+AC04                 DW    (INQCB$ARRAY + ??XX)
0068+6A00                 DW    $+2
006A+0000                 DW    0
006C+C604                 DW    (INQCB$ARRAY + ??XX)
006E+7000                 DW    $+2
0070+0000                 DW    0

                ;       UQCB used by ServeRX to get free buffers from Q
0072 1E057600   gbuf$uqcb:      dw      buffQCB,newbuff
0076 0000       newbuff:        dw      0               ;message is a free buffer ptr from pool

                ;       UQCB used by ServeRX to get transporter from MX Q
0078 A8087C00   omnirx$uqcb:    dw      omniQ,rx$mx$msg
007C 0000       rx$mx$msg:      dw      0

                ;       UQCB used by ServeRX to send error messages to outQ
007E E0048200   err$out$uqcb:   dw      outQCB,err$out$msg ;pointer, msgadr
0082            err$out$msg:    ds      2                  ;used to send error messages

                ;       receiver transporter control block
0084 F0         rxtcb:  db      0f0h                    ;post read command
0085 00                 db      0                       ;result hi (always 0)
                rxrsltp:
0086 0000               db      0,0                     ;result middle and low (NOT 8080 order)
0088 A0                 db      OMNI$SOCKET             ;transporter message socket code
0089 00                 db      0                       ;data pointer high (always 0)
008A 0000               db      0,0                     ;data pointer middle, low
008C 01                 db      BUFFSIZE/256            ;data max length hi
008D 18                 db      BUFFSIZE and 255        ;data max length lo
008E 0000               db      0,0                     ;ctrl lgth (0 for now), host (not used)

0090 0000000000rxrslt: db      0,0,0,0,0,0,0,0         ;result block for rx

                ;_______________________________________________________________________
                ;                                                                       ;
                ;       Receiver server process initialization entry point              ;
```

```
                    ;               (initializes all of module)                           ;
                    ;_______________________________________________________________________
0098 CDCD08         InitRX: call    omni$init       ;init hardware & get ID code from its switches
009B 32FB02                 sta     configtbl+1     ; store ID in config table as master ID
                    ;
009E 0E86                   mvi     c,MAKEQ         ;create the free buffer Q
00A0 111E05                 lxi     d,buffQCB
00A3 CDA408                 call    bdos
                    ;
00A6 11AC04                 lxi     d,inqcb$array
00A9 0E02                   mvi     c,NSLAVES       ;create input Qs (1/slave supported)
                    make$inQs:
00AB D5                     push    d
00AC C5                     push    b
00AD 0E86                   mvi     c,MAKEQ
00AF CDA408                 call    bdos
00B2 C1                     pop     b
```



```
00B3 D1                     pop     d
00B4 211A00                 lxi     h,INQCB$SIZE
00B7 19                     dad     d
00B8 EB                     xchg
00B9 0D                     dcr     c
00BA C2AB00                 jnz     make$inQs
                    ;
00BD 11E004                 lxi     d,outQCB        ;create the output Queue (only 1)
00C0 0E86                   mvi     c,MAKEQ
00C2 CDA408                 call    bdos
                    ;
00C5 11B901                 lxi     d,ServetxPD     ;create the network output process
00C8 0E90                   mvi     c,CREATEP
00CA CDA408                 call    bdos
                    ;
00CD 0E9A                   mvi     c,SYDATAD       ;get system data page address
00CF CDA408                 call    bdos
00D2 110900                 lxi     d,9
00D5 19                     dad     d               ;install config table address at sysdat(9)
00D6 11FA02                 lxi     d,configtbl
00D9 73                     mov     m,e
00DA 23                     inx     h
00DB 72                     mov     m,d
                    ;
00DC 219000                 lxi     h,rxrslt        ;initialize transporter command block result
00DF 55                     mov     d,l             ;field to point to receive result block
00E0 5C                     mov     e,h             ; (done at run time because of reversed byte
00E1 EB                     xchg                    ;  order used by CORVUS.)
00E2 228600                 shld    rxrsltp


                    ;       Receiver server process loop head
00E5 0E89           RXloop: mvi     c,READQ
00E7 117200                 lxi     d,gbuf$uqcb
00EA CDA408                 call    bdos            ;get a free message buffer from Q
                    ;
                    RXretry:
00ED 2A7600                 lhld    newbuff
00F0 5C                     mov     e,h
00F1 55                     mov     d,l
00F2 EB                     xchg                    ;swap bytes for CORVUS command block
00F3 228A00                 shld    rxtcb+6         ;put buffer address pointer in rx tcb
                    ;
00F6 117800                 lxi     d,omnirx$uqcb   ;read MX message from OMNINET HARDWARE MX Q
00F9 0E89                   mvi     c,READQ
00FB CDA408                 call    bdos
                    ;
00FE 018400                 lxi     b,rxtcb         ;send TCB pointer to hardware
0101 CDF508                 call    omni$strobe
                    ;
0104 F5                     push    psw             ;return MX message
0105 117800                 lxi     d,omnirx$uqcb
0108 0E8B                   mvi     c,WRITEQ
010A CDA408                 call    bdos
```

```
010D F1                 pop     psw             ;restore return code from omni$strobe routine
                ;
010E DAED00             jc      RXretry         ;no choice except to retry if not accepted
                ;
0111 019000             lxi     b,rxrslt        ;wait for a completion from hardware
0114 CD2309             call    wfrxdone
0117 E680               ani     80h             ;if error on message, re-post buffer
0119 C2ED00             jnz     RXretry
                ;
                ;  buffer contains a valid message at this point, so process it
011C 2A7600             lhld    newbuff         ;get FMT to A
011F 7E                 mov     a,m
0120 23                 inx     h               ;get SID to C
0121 23                 inx     h
0122 4E                 mov     c,m
                ;
0123 E6FE               ani     0feh            ;look for login/logoff messages
0125 C24601             jnz     RXl2            ;message type 0 or 1?
0128 23                 inx     h               ;yes, check FNC
0129 7E                 mov     a,m
012A FE40               cpi     40h             ;login?
012C C23801             jnz     RXl1            ;not login, go on
012F CDA301             call    logiton         ;ELSE try to find a free SLVSP in table
0132 C26C01             jnz     RXl3            ;found one (or already logged in), go on
0135 C34C01             jmp     RX$send$err     ;sorry,no free processes, go advise slave
                ;
0138 FE41       RXl1:   cpi     41h             ;logoff?
013A C24601             jnz     RXl2            ;not logoff, go on
013D CD9A01             call    logitoff        ;ELSE try to remove that slave from table
0140 C26C01             jnz     RXl3            ;if successful, go on
0143 C34C01             jmp     RX$send$err     ;otherwise go tell slave it wasn't logged in
                ;
0146 CD8001     RXl2:   call    get$slvsp       ;not login/logoff so get slvsp msg address
0149 C26C01             jnz     RXl3            ; for that slave if it is logged in and go
                                                ; send message to its Q else fall through
                ;
                ;       this code sends a "NOT LOGGED IN" error message back to requester
                RX$send$err:
014C 2A7600             lhld    newbuff         ;build an error message in the same buffer
014F 228200             shld    err$out$msg
0152 3601               mvi     m,1             ;FMT=1
0154 23                 inx     h
0155 7E                 mov     a,m             ;swap DID and SID
0156 23                 inx     h
0157 46                 mov     b,m
0158 77                 mov     m,a
0159 2B                 dcx     h
015A 70                 mov     m,b
015B 23                 inx     h               ;leave FNC field alone
015C 23                 inx     h
015D 23                 inx     h
015E 3601               mvi     m,1             ;SIZ=1
0160 23                 inx     h
0161 36FF               mvi     m,0ffh          ;message = 0FFH (extended error flag)
```



```
0163 23                 inx     h
0164 360C               mvi     m,12            ;"NOT LOGGED IN" code
0166 117E00             lxi     d,err$out$uqcb  ;post to network transmitter process
0169 C37801             jmp rxl4                ;using common write Q code
                ;
                ;  this code sends the message address to the appropriate SLVSP Q
016C 2A7600     RXl3:   lhld    newbuff         ;DE--> msg field of correct UQCB here
016F EB                 xchg                    ;put message ptr in UQCB message field
0170 73                 mov     m,e
0171 23                 inx     h
0172 72                 mov     m,d
0173 11FBFF             lxi     d,-(XQCBMSG + 1);index back to UQCB base address
0176 19                 dad     d
0177 EB                 xchg
                ;
0178 0E8B       rxl4:   mvi     c,WRITEQ
017A CDA408             call    bdos            ;send it to Queue
017D C3E500             jmp     RXloop          ;go back and get another buffer and continue
```

```
                ;       routine dynamically maps physical slave number passed in C
                ;       to a slave support process and returns its INUQCB message buffer addr
                ;       in DE and A = 0 with flags set if no room or not found, else NZ
                get$slvsp:
0180 79                 mov     a,c             ;A= requester ID
0181 0602               mvi     b,NSLAVES       ;set up for table search
0183 21B301             lxi     h,idtbl
                find$match:                     ;search till match or table end
0186 BE                 cmp     m
0187 C29101             jnz     not$match       ; goto not$match if not this one
018A 23                 inx     h               ;else match found, get ptr to SLVSP message
018B 5E                 mov     e,m
018C 23                 inx     h
018D 56                 mov     d,m             ;its slvsp msg addr
018E 37                 stc
018F 9F                 sbb     a
0190 C9                 ret                     ;and return TRUE in A to caller
                not$match:
0191 23                 inx     h               ;no match, skip to next entry
0192 23                 inx     h
0193 23                 inx     h
0194 05                 dcr     b               ;any more entries?
0195 C28601             jnz     find$match      ;loop back until all searched
0198 AF                 xra     a               ;else return failure (A=00)
0199 C9                 ret


                ;       removes entry (C=SID) from map table (but still returns msg ptr)
                logitoff:
019A CD8001             call    get$slvsp
019D C8                 rz                      ;not in table, just exit
019E 2B                 dcx     h               ;else mark entry as free and then exit
019F 2B                 dcx     h
01A0 36FF               mvi     m,0ffh
```



```
01A2 C9                 ret

                ;       installs entry (C=SID) in first free entry of map table and returns
                ;       msg address. RETURNS A=0 if no space, else non-zero.
                logiton:
01A3 CD8001             call    get$slvsp       ;see if already in table
01A6 C0                 rnz                     ;if so, just use old entry
01A7 C5                 push    b               ;else look for a free entry (CODE=FF)
01A8 0EFF               mvi     c,0ffh
01AA CD8001             call    get$slvsp
01AD C1                 pop     b
01AE C8                 rz                      ;no free entries, exit
01AF 2B                 dcx     h               ;else enter SID in table and return success
01B0 2B                 dcx     h
01B1 71                 mov     m,c
01B2 C9                 ret                     ;PSW is still correct from search

                ;       Slave mapping table has one entry per SLVSP.  First  byte = SID
                ;       of the requester currently using SLVSP (0ffh if none).  Next word is
                ;       the address of the message field of that SLVSP's input UQCB.
                idtbl:
0000 #          ??xx    set     0
                        rept    NSLAVES
                          db    0ffh
                          dw    (INUQCB + XQCBMSG + ??xx)
                ??xx      set   ??xx + UQCBLEN
                        endm
01B3+FF                   DB    0FFH
01B4+6A00                 DW    (INUQCB + XQCBMSG + ??XX)
01B6+FF                   DB    0FFH
01B7+7000                 DW    (INUQCB + XQCBMSG + ??XX)

                page
```



```
                ;_________________________________________________________________________
```

```
                    ;__________________________________________________________________________
                    ;                                                                         ;
                    ;       This is the network transmitter server process module.            ;
                    ;       NOTE THAT THE OMNINET TRANSPORTER MUST NOT BE DISTURBED ONCE      ;
                    ;       A TRANSMIT HAS BEEN POSTED UNTIL IT RETURNS A COMPLETION.         ;
                    ;       An MX Queue is used in this version to protect the transporter    ;
                    ;       from other processes.                                             ;
                    ;                                                                         ;
                    ;       This process reads a message from the SLVSP output Q and when     ;
                    ;       awakened by one posts that buffer for transmission via the        ;
                    ;       transporter to the requester.  This process then waits until      ;
                    ;       the transporter reports a completion as determined by the         ;
                    ;       wf$txdone routine.  The buffer pointer from that message is       ;
                    ;       then sent back to the FreeBuff Q and the process loops back for   ;
                    ;       another message from the SLVSP output Q.  Transmitter errors      ;
                    ;       are considered the Transporter's problem and are ignored here.    ;
                    ;__________________________________________________________________________
                    ;__________________________________________________________________________
                    ;       Transmitter server process descriptor
                    ServetxPD:
      01B9 0000             dw      0                       ;link
      01BB 003F             db      0,TX$PRIORITY           ;status,priority
      01BD 1B02             dw      $ + 94                  ;stack pointer
      01BF 5365727665       db      'ServeTX '              ;name
      01C7 00FF             db      0,0ffh                  ;console, memseg
      01C9                  ds      82                      ;reserved for MP/M use and as stack
      021B 4302             dw      InitTX                  ;stack top has startup PC

                    ;       There is only one output queue (SLVSP --> NTWRKIF)
                    OUTUQCB:
      021D E0042102 UQCBNtwrkQO0:   dw      outQCB,outQMSG  ;pointer, msgadr
      0221          outQMSG:        ds      2               ;used to receive msg pointer from SLVSP

                    ;       used by ServeTX to return them to Q when done (used at init also)
      0223 1E052702 pbuf$uqcb:      dw      buffQCB,oldbuff
      0227 0000     oldbuff:        dw      0               ;msg is a freed buff ptr back to pool

                    ;       UQCB used by ServeTX to get transporter from MX Q
      0229 A8082D02 omnitx$uqcb:    dw      omniQ,tx$mx$msg
      022D 0000     tx$mx$msg:      dw      0

                    ;       transmitter transporter control block
      022F 40       txtcb:  db      40h                     ;command
      0230 00               db      0                       ;result hi
                    txrsltp:
      0231 0000             db      0,0                     ;result middle and low
      0233 A0               db      OMNI$SOCKET             ;transporter message socket code
      0234 000000           db      0,0,0                   ;data ptr (MSB,SB,LSB)
      0237 0000             db      0,0                     ;length (MSB,LSB)
      0239 00               db      0                       ;control length
      023A 00               db      0                       ;dest host
```



```
      023B 0000000000txrslt: db     0,0,0,0,0,0,0,0         ;result block for tx

                    ;__________________________________________________________________________
                    ;                                                                         ;
                    ;       ServeTX initialization entry point                                ;
                    ;__________________________________________________________________________
                    InitTX:
      0243 215C05           lxi     h,msgbuffs      ;preload the Free buffer Q with buffer ptrs
      0246 0E03             mvi     c,NMSG$BUFFS    ;from start of buffer space
                    freeloop:
      0248 222702           shld    oldbuff
      024B E5               push    h
      024C C5               push    b
      024D 0E8B             mvi     c,WRITEQ
      024F 112302           lxi     d,pbuf$uqcb
      0252 CDA408           call    bdos
      0255 C1               pop     b
      0256 E1               pop     h
      0257 111801           lxi     d,BUFFSIZE
      025A 19               dad     d
      025B 0D               dcr     c
      025C C24802           jnz     freeloop
```

```
                        ;
        025F 213B02             lxi     h,txrslt        ;initialize TX Transporter Command Block
        0262 5C                 mov     e,h             ;to point to TX Result Block
        0263 55                 mov     d,l
        0264 EB                 xchg
        0265 223102             shld    txrsltp

                        ;       ServeTX process loop
                        TXloop:
        0268 0E89               mvi     c,READQ         ;wait for a message in network output Q
        026A 111D02             lxi     d,outuqcb
        026D CDA408             call    bdos
                        ;
        0270 2A2102             lhld    outQMSG
        0273 5C                 mov     e,h
        0274 55                 mov     d,l             ;put message buffer address in TX TCB
        0275 EB                 xchg                    ;(NOTE, NOT (8080 byte order)
        0276 223502             shld    txtcb+6
                        ;
        0279 13                 inx     d
        027A 1A                 ldax    d               ;set transport layer destination addr=DID
        027B 323A02             sta     txtcb + 11
                        ;
        027E 210300             lxi     h,3
        0281 19                 dad     d               ;calculate physical message length
        0282 6E                 mov     l,m             ;from SIZ field
        0283 2600               mvi     h,0
        0285 110600             lxi     d,6             ;put in TCB length field
        0288 19                 dad     d
        0289 55                 mov     d,l
        028A 5C                 mov     e,h
        028B EB                 xchg
```



```
        028C 223702             shld    txtcb+8
                        ;
        028F 112902             lxi     d,omnitx$uqcb   ;get transporter hardware MX message
        0292 0E89               mvi     c,READQ
        0294 CDA408             call    bdos
                        ;
                        TXretry:
        0297 012F02             lxi     b,txtcb         ;send TCB pointer to hardware
        029A CDF508             call    omni$strobe     ;if can't, not much else to do but try again
        029D DA9702             jc      TXretry         ;  (ALTHOUGH THIS IS A FATAL HARDWARE ERROR)
                        ;
        02A0 013B02             lxi     b,txrslt        ;wait for transmit completion
        02A3 CD3409             call    wftxdone        ;ignore errors here as no recovery possible
                        ;
        02A6 112902             lxi     d,omnitx$uqcb
        02A9 0E8B               mvi     c,WRITEQ
        02AB CDA408             call    bdos            ;release MX msg
                        ;
        02AE 2A2102             lhld    outQMSG         ;send the buffer back to FREEBUFF Q
        02B1 222702             shld    oldbuff
        02B4 0E8B               mvi     c,WRITEQ
        02B6 112302             lxi     d,pbuf$uqcb
        02B9 CDA408             call    bdos
                        ;
        02BC C36802             jmp     txloop          ;and go back and do it all with next msg


                        page
```



```

        02BF 4E5457524Bcnote:  db 'NTWRKIF (c)1982 VANO ASSOCIATES, INC. - ALL RIGHTS RESERVED'
                        ;_________________________________________________________________________
                        ;_________________________________________________________________________
                        ;                                                                        ;
                        ;       GLOBAL Master Configuration Table and storage                    ;
                        ;       (address must be installed on SysData page(9,10) at init.)       ;
                        ;_________________________________________________________________________
                        ;_________________________________________________________________________
                        configtbl:
```

```
02FA 00               db      0               ;Master status byte
02FB 00               db      0               ;Master processor ID
02FC 02               db      NSLAVES         ;Maximum number of slaves supported
02FD 00               db      0               ;Number of logged in slaves
02FE 0000             dw      0               ;16 bit vector of logged in slaves
0300                  ds      16              ;Slave processor ID array
0310 5041535357       db      'PASSWORD'      ;login password

              ;       builds Server stacks and initializes them with PD storage pointers
0000 #        ??xx    set     0
                      rept    NSLAVES
                        ds    SRVR$STK$SIZ - 2
                        dw    srvr$pd$base + ??xx
                        ??xx  set ??xx + SRVR$PD$SIZ
                      endm
0318+                   DS    SRVR$STK$SIZ - 2
03AC+4404               DW    SRVR$PD$BASE + ??XX
03AE+                   DS    SRVR$STK$SIZ - 2
0442+7804               DW    SRVR$PD$BASE + ??XX

              ;       allocates PD storage
              srvr$pd$base:
0444                  ds      NSLAVES * SRVR$PD$SIZ

              ;_______________________________________________________________________
              ;_______________________________________________________________________
              ;                                                                       ;
              ;       INTERPROCESS QUEUES (both local and global) and COMMON data     ;
              ;_______________________________________________________________________
              ;_______________________________________________________________________

              ;       ServeRX --> SLVSP message queues (INPUT), 1/slave support proc.
001A =        INQCB$SIZE      equ     26      ;constant used for index calculation
              inqcb$array:                    ;ARRAY BASE NAME
              ;
              ;       generate INQCBs as required
0030 #        ??xx    set     '0'
                      rept    NSLAVES
                        ds    2               ;;link
                        db    4eh,74h,77h,72h ;;common name is NTwrkQI
                        db    6bh,51h,49h     ;;(macro can't do lower case)
                        db    ??xx            ;;slave ID
                        dw    2,1             ;;msglen, nmbmsgs
                        ds    12              ;;MP/M pointers and buffers
```



```
              ??xx      set   ??xx + 1
                if (??xx EQ ('9'+1))
              ??xx        set ??xx + 7
                endif
                      endm
04AC+                   DS    2
04AE+4E747772           DB    4EH,74H,77H,72H
04B2+6B5149             DB    6BH,51H,49H
04B5+30                 DB    ??XX
04B6+02000100           DW    2,1
04BA+                   DS    12
04C6+                   DS    2
04C8+4E747772           DB    4EH,74H,77H,72H
04CC+6B5149             DB    6BH,51H,49H
04CF+31                 DB    ??XX
04D0+02000100           DW    2,1
04D4+                   DS    12

              ;       SLVSP --> NETWRKIF queue (OUTPUT)
04E0          outQCB: ds      2               ;link
04E2 4E7477726B       db      'NtwrkQO0'      ;name
04EA 02001000         dw      2,16            ;msglen, nmbmsgs
04EE                  ds      48              ;Used by MP/M

              ;       free buffer list management queue
              buffQCB:
051E                  ds      2               ;link
0520 4672656542       db      'FreeBuff'      ;name
0528 02001000         dw      2,16            ;msglen, nmbmsgs
052C                  ds      48              ;reserved for MP/M
```

```


                ;       global message buffer pool
055C            msgbuffs:       ds      NMSG$BUFFS * BUFFSIZE

                ;       Utility Procedure to allow indirect BDOS/XDOS access as needed by RSP
08A4 2A0000     bdos:   lhld    bdosadr
08A7 E9                 pchl

                page
```



```

                ;________________________________________________________________________
                ;________________________________________________________________________
                ;                                                                       ;
                ;       low level omninet support routines                              ;
                ;________________________________________________________________________
                ;________________________________________________________________________

                ;       Transporter mutual exclusion QUEUE
08A8            omniQ:  ds      2
08AA 4D586F6D6E         db      'MXomniQ '
08B2 00000100           dw      0,1                     ;msglen, nmsgs
08B6                    ds      12                      ;dqph,nqph,msgin,msgout,msgcnt,buff

                ;       UQCB used by omni$init to load MX Q
08C2 A808C608   omni$init$uqcb: dw      omniQ,init$mx$msg
08C6 0000       init$mx$msg:    dw      0


                ;       Initialization transporter control block
                inittcb:
08C8 20                 db      20h                     ;command
08C9 00                 db      0                       ;result hi
                initrsltp:
08CA 0000               db      0,0                     ;result middle and low
                ;
                initrslt:
08CC 00                 db      0                       ;result block for init


                ;       initializes transporter hardware and return its network ID code in A
                omni$init:
08CD 11A808             lxi     d,omniQ
08D0 0E86               mvi     c,MAKEQ
08D2 CDA408             call    bdos            ;create hardware MX Q
08D5 11C208             lxi     d,omni$init$uqcb ;send it one message
08D8 0E8B               mvi     c,WRITEQ
08DA CDA408             call bdos
                  if INTERRUPT
                        call int$init           ;(optional) setup interrupt system
                  endif
08DD 21CC08             lxi     h,initrslt      ;install result block pointer in initialization
08E0 55                 mov     d,l             ;TCB
08E1 5C                 mov     e,h             ;NOTE: NOT 8080 order, MSB,LSB
08E2 EB                 xchg
08E3 22CA08             shld    initrsltp
                ;
08E6 01C808             lxi     b,inittcb       ;post initialization command block to
08E9 CDF508             call    omnistrobe      ;hardware
08EC D8                 rc                      ;cy=1 means can't talk to hardware
                ;
08ED 01CC08             lxi     b,initrslt      ;wait for a completion from operation
08F0 CD2309             call    omni$wfdone
08F3 B7                 ora     a
```



```
08F4 C9                 ret                     ;return ID/result code to caller with flags set


                ;       sends the command block pointer in BC to transporter hardware
                omni$strobe:
08F5 210200             lxi     h,2             ;first preset result code byte in
```

```
08F8 09                 dad     b               ;result block TCB result field --> to 0ffh
08F9 7E                 mov     a,m
08FA 23                 inx     h
08FB 6E                 mov     l,m
08FC 67                 mov     h,a
08FD 36FF               mvi     m,0ffh
                ;
08FF AF                 xra     a               ;send bits 23-16 of ptr to hardware (always 0)
0900 CD0A09             call    omni$st
0903 D8                 rc                      ;carry means can't talk to hardware
                ;
0904 78                 mov     a,b             ;send bits 15-8 of ptr to hardware
0905 CD0A09             call    omni$st
0908 D8                 rc
                ;
0909 79                 mov     a,c             ;send bits 7-0 of ptr to hardware
                                                ;fall into omni$st to send last byte

                ;       called by omni$strobe to send one byte to transporter when ready
                ;       (waits a reasonable time for transporter to come ready and if
                ;       it doesn't, returns with carry set;  this is a fatal error) returns
                ;       cy=0 if succeeds
                omni$st:
090A F5                 push    psw             ;save data for now
090B 1150C3             lxi     d,50000         ;set timeout
                omni$st0:
090E DBF9               in      OMNI$STAT       ;see if transporter will accept byte
0910 E610               ani     OMNI$RDY
0912 CA1A09             jz      omni$st1        ;if busy, go decrement timeout and retry
0915 F1                 pop     psw             ;else output the byte and return with CY=0
0916 D3F8               out     OMNI$DATA
0918 B7                 ora     a
0919 C9                 ret
                omni$st1:
091A 1B                 dcx     d               ;loop back if not timeout yet
091B 7B                 mov     a,e
091C B2                 ora     d
091D C20E09             jnz     omni$st0
0920 F1                 pop     psw
0921 37                 stc
0922 C9                 ret                     ;else return CY=1 as error flag


                ;       routine waits for a completion to occur on the result block
                ;       pointed to by BC.  This routine is used by the initialization
                ;       and receiver processes.  If there is no interrupt hardware in
                ;       the system, ONLY ONE MESSAGE CAN BE RECEIVED PER CLOCK TICK of
                ;       the system clock.  This will considerably reduce server throughput
```



```
                ;       in most systems.
                omni$wfdone:
                wfrxdone:
0923 0A                 ldax    b               ;all completion codes are < 0f0h
0924 FEF0               cpi     0f0h            ;see if already done before suspending caller
0926 D8                 rc                      ;yes, return immediately
                ;       else suspend caller until a completion occurs
0927 C5                 push    b
                  if INTERRUPT
                        lxi     d,OMNI$FLAG     ;wait for ISR to set flag
                        mvi     c,FLAGWAITF
                        call    bdos
                  else
0928 110100             lxi     d,1             ;if no ISR, poll result block once/tick
092B 0E8D               mvi     c,DELAY
092D CDA408             call    bdos
                  endif
0930 C1                 pop     b
0931 C32309             jmp     omni$wfdone

                ;       As above but instead polls continually to give transmitter priority
                ;       since transmitter usually unloads messages in less time than MP/M
                ;       dispatch overhead, it is not worth suspending it.
                ;       A timeout routine is included to avoid locking up system if hardware
                ;       fails so diagnosing the problem is possible with RDT.
                wftxdone:
```

```
0934 1150C3           lxi     d,50000         ;initialize hardware fail timeout
0937 0A       wftxd0: ldax    b               ;done yet?
0938 FEF0             cpi     0f0h
093A 3F               cmc                     ;set up carry properly in case of return
093B D0               rnc                     ;yes, return to caller with result in A, CY=0
093C 1B       wftxd1: dcx     d               ;if not timeout, loop back
093D 7B               mov     a,e
093E B2               ora     d
093F C23709           jnz     wftxd0
0942 37               stc
0943 C9               ret                     ;else return to caller with CY=1 as error flag

              page
```



```

                if INTERRUPT
              ;
              ;       Since the CORVUS "ENGINEERING" transporter has no interrupt hardware
              ;       associated with it, the details of the interrupt initialization and
              ;       service routines will vary from system to system.  The skeleton of
              ;       our code is provided here as a guide to understanding what is needed.
              ;
              ;       Routine initializes interrupt hardware and attaches ISR to XIOS
              ;       at run-time (in somewhat bizarre fashion.)  It would be better
              ;       to make your ISR a permanent part of your XIOS since if not
              ;       used it does no harm to the system.
              int$init:
                      di
                      mvi     a,(jmp)         ;build jump in vector
                      sta     (INT$VCTR)
                      lxi     h,omni$isr
                      shld    (INT$VCTR + 1)  ;install new isr
                      out     OMNI$ACK        ;clear interrupt latch
                      mvi     a,OMNI$ENABLE   ;unmask transporter interrupt
                      out     OMNI$MASK
              ; this code does an extremely Klugey run-time linkage to needed XIOS routines
                      lhld    1               ;find CBOOT in MPM-II BIOS simulation table
                      mvi     l,1
                      mov     e,m
                      inx     h
                      mov     d,m
                      push    d               ;save to find exit$reg.
              ;
                      xchg                    ;need to go one more level to find real entry
                      inx     h
                      mov     e,m
                      inx     h
                      mov     d,m             ;this is address of real CBOOT entry in XIOS
              ;
                      lxi     h,9             ;calculate PDISP entry from CBOOT address
                      dad     d
                      shld    pdisp           ;and save it in local vector
              ;
                      lxi     d,3             ;XDOS address is 3 bytes above PDISP
                      dad     d
                      shld    xd$adr          ;save it in a local vector
              ;
                      pop     h               ;get XIOS branch table address back
                      mvi     l,40h           ;calculate address of EXIT$REGION entry
                      mov     e,m
                      inx     h
                      mov     d,m
                      xchg
                      shld    exit$region     ;save it for later use in pre-empt routine
                      ei
                      ret

              ;       omninet isr sets the appropriate XDOS flag and causes a dispatch
```



```
              omni$isr:
                      shld    svhl
                      pop     h
```

```
 864                    push    psw             ;save PSW and HL
 865                    shld    svret           ;save return address
 866                    lxi     h,0             ;swap stacks
 867                    dad     sp
 868                    shld    svstk
 869                    lxi     sp,isr$stk
 870                    push    d               ;save the other registers on new stack
 871                    push    b
 872            ;
 873                    out     OMNI$ACK        ;clear interrupt latch
 874            ;
 875                    lhld    exit$region     ; do a PRE-EMPT by patching a RET into table
 876                    mov     a,m             ; (Very KLUGEY but there's no other way.)
 877                    push    psw             ; save what was in XIOS branch table entry
 878                    push    h               ; and put a RET there to prevent XDOS from
 879                    mvi     m,(RET)         ; re-enabling interrupts
 880            ;
 881                    mvi     c,FLAGSETF      ;call XDOS to set isr flag
 882                    mvi     e,OMNI$FLAG
 883                    call    xdos
 884            ;
 885                    pop     h
 886                    pop     psw
 887                    mov     m,a             ;restore XIOS table entry
 888            ;
 889                    pop     b               ;pop interrupted registers
 890                    pop     d
 891                    lhld    svstk           ;restore interrupted stack
 892                    sphl                    ;restore other regs. and exit
 893                    pop     psw
 894                    lhld    svret
 895                    push    h
 896                    lhld    svhl
 897                    db      (JMP)           ; via dispatcher
 898            pdisp:  dw      0               ;(link to dispatcher)
 899
 900            xdos:   db      (JMP)           ;special XDOS entry
 901            xd$adr: dw      0               ;for ISR use
 902
 903            ;       ISR data areas
 904            exit$region:
 905                    dw      0               ;address of XDOS critical region exit routine
 906                    ds      64              ;isr stack space
 907            isr$stk:
 908            svhl:   dw      0               ;temporary reg storage
 909            svret:  dw      0
 910            svstk:  dw      0               ;careful, make sure all of .RSP is reserved
 911
 912              endif ; of if INTERRUPT
 913
 914   0944             end
```



```
BDOS             08A4   110   113   225   233   244   248   251   269   280   288
                        359   499   517   542   554   560   656#  694   697   779
                        783
BDOSADR          0000   117#  122#  656
BUFFQCB          051E   190   224   466   645#
BUFFSIZE         0118    62#  209   210   502   653
CNOTE            02BF   567#
CODESEG          0000   104#  118
CONFIGTBL        02FA   221   254   575#
CR               000D   100#
CREATEP          0090    94#  247
DEBUG            0000    54#  111
DELAY            008D    92#  782
DETACH           0093    96#  112
DSPTCH           008E    93#
ERROUTMSG        0082   198   199#  328
ERROUTUQCB       007E   198#  345
FINDMATCH        0186   370#  385
FLAGSETF         0085    88#  881
FLAGWAITF        0084    87#  778
FREELOOP         0248   493#  505
GBUFUQCB         0072   190#  268
GETSLVSP         0180   321   366#  392   402   406
```

```
IDTBL            01B3   369   417#
INITMXMSG        08C6   675   676#
INITRSLT         08CC   686#  701   711
INITRSLTP        08CA   683#  705
INITRX           0098   167   220#
INITTCB          08C8   680#  707
INITTX           0243   458   490#
INQCBARRAY       04AC   177   182   185   227   609#
INQCBSIZE        001A   180   236   608#
INTERRUPT        0000    56#  698   776   808
INTVCTR          0038    72#  822   824
INUQCB           0066   174#  421   425   427
LF               000A   101#
LOGITOFF         019A   317   391#
LOGITON          01A3   311   401#
MAKEINQS         00AB   229#  240
MAKEQ            0086    89#  223   232   243   693
MSGBUFFS         055C   491   653#
NEWBUFF          0076   190   191#  272   299   327   349
NMSGBUFFS        0003    63#  492   653
NO               0000    51#   54    56
NOTMATCH         0191   372   380#
NSLAVES          0002    59#   63   176   228   368   419   578   586   598   613
OLDBUFF          0227   466   467#  494   557
OMNIACK          00FA    79#  825   873
OMNIBASE         00F8    68#   75    76    79    80
OMNIDATA         00F8    75#  750
OMNIDISABLE      0000    83#
OMNIENABLE       0001    82#  826
OMNIFLAG         0008    70#  777   882
OMNIINIT         08CD   220   691#
```



```
OMNIINITUQCB     08C2   675#  695
OMNIMASK         00FB    80#  827
OMNIPENDING      0001    81#
OMNIQ            08A8   194   470   669#  675   692
OMNIRDY          0010    77#  747
OMNIRXUQCB       0078   194#  278   286
OMNISOCKET       00A0    69#  206   478
OMNIST           090A   728   732   742#
OMNIST0          090E   745#  757
OMNIST1          091A   748   753#
OMNISTAT         00F9    76#  746
OMNISTROBE       08F5   283   546   708   718#
OMNITXUQCB       0229   470#  540   552
OMNIWFDONE       0923   712   769#  786
OUTQCB           04E0   198   242   462   639#
OUTQMSG          0221   462   463#  519   556
OUTUQCB          021D   461#  516
PBUFUQCB         0223   466#  498   559
PRINTF           0009    86#
READQ            0089    90#  267   279   515   541
RSP              FFFF    55#  105
RSTNUM           0007    71#   72
RXL1             0138   310   315#
RXL2             0146   306   316   321#
RXL3             016C   312   318   322   349#
RXL4             0178   346   358#
RXLOOP           00E5   267#  360
RXMXMSG          007C   194   195#
RXPRIORITY       0040    64#  109   161
RXRETRY          00ED   271#  291   296
RXRSLT           0090   213#  259   293
RXRSLTP          0086   204#  263
RXSENDERR        014C   313   319   326#
RXTCB            0084   202#  276   282
SERVERXPD        0002   159#
SERVERXSTKTOP    0064   107   166#
SERVETXPD        01B9   246   451#
SETPRIORITY      0091    95#  108
SRVRPDBASE       0444   588   592   594   597#
SRVRPDSIZ        0034    61#  589   598
SRVRSTKSIZ       0096    60#  587   591   593
SYDATAD          009A    97#  250
TXLOOP           0268   514#  562
```

```
TXMXMSG          022D   470   471#
TXPRIORITY       003F    65#  453
TXRETRY          0297   544#  547
TXRSLT           023B   484#  507   549
TXRSLTP          0231   476#  511
TXTCB            022F   474#  523   527   538   545
UQCBLEN          0006   171#  422
UQCBNTWRKQO0     021D   462#
WFRXDONE         0923   294   770#
WFTXD0           0937   795#  802
WFTXD1           093C   799#
```

```
CP/M RMAC ASSEM 1.1     #022    SAMPLE SERVER NETWORK I/F FOR CORVUS OMNINET 20-OCT-82

WFTXDONE         0934   550   793#
WRITEQ           008B    91#  287   358   497   553   558   696
XQCBMSG          0004   172#  354   421   425   427
YES              FFFF    50#   51    55
```

Listing G-2. Sample Server Network I/O for Corvus OMNINET