AN ABSTRACT OF THE THESIS OF

Harish Pillay for the degree of Master of Science in Electrical and Computer Engineering presented on March 29, 1990.

Title: An Implementation of the Department of Defense's Transmission Control Protocol/Internet Protocol (TCP/IP) for the Microsoft OS/2 Operating System

Redacted for privacy

Abstract approved: John M. Murray

This thesis discusses an approach whereby Microsoft's MS OS/2 is provided with a means of running the Department of Defense's Transmission Control Protocol/Internet Protocol (TCP/IP).

This is done by developing a Packet Protocol Device Driver. This device driver complies with the Packet Driver Specification from FTP Software Inc. and with the Network Driver Interface Specification (NDIS) from 3Com and Microsoft Corporations. This packet protocol device driver co-resides with other protocol device drivers and shares one medium access control (MAC) device driver as defined in the NDIS.

With the successful implementation of the packet protocol device driver, an existing Microsoft MS-DOS version of a TCP/IP package was ported and with minor modifications recompiled to run under MS OS/2. This method allows users to retain utility and use of the OS/2 LAN Manager, a networking strategy provided within MS OS/2.

An Implementation of the Department of Defense's Transmission Control Protocol/ Internet Protocol (TCP/IP) for the Microsoft OS/2 Operating System

by

Harish Pillay

A THESIS

submitted to

Oregon State University

in partial fulfillment of the requirements for the degree of

Master of Science

Completed March 29, 1990

Commencement June 1990

APPROVED:

Redacted for privacy

Associate Professor of Electrical and Computer Engineering in charge of major

Redacted for privacy

Head of Department of Electrical and Computer Engineering

Redacted for privacy

Dean of Graduate School

Date Thesis is presented March 29, 1990

Type by Harish Pillay for Harish Pillay

Acknowledgement

I would like to acknowledge all the help and advise given to me by my major professor, Prof. John Murray. His wit and humor was especially welcome. I would also like to thank the members of my graduate committee, Prof. Bruce D'Ambrosio and Prof. James Herzog for all their assistance.

A lot of suggestions and hints came from a wide variety of sources too numerous to mention here. However, the one source that I would like to specifically mention is USENET. USENET's newsgroups such as comp.protocols.tcp-ip, comp.os.os2, and comp.sys.ibm.pc were just wonderful in generating ideas and solutions.

There are some very special people who I would like to record my gratitude. They are my mom and dad, Santha and Balakrishna Pillay, and my sister and brother-in-law, Srila and Venugopal Kurup who extended their love, understanding, encouragement and concern throughout this venture.

And finally, I would like to dedicate of all of this work to my nephew, Jeevan, who will grow up in a world full of peace, understanding, and goodwill achieved through the wonders of networking.

Table of Contents

List of Figures

List of Tables

An Implementation of the Department of Defense's Transmission Control Protocol/Internet Protocol (TCP/IP) for the Microsoft OS/2 Operating System

Chapter 1 Objectives

1.1 Introduction

The goal of the research and development effort of this thesis is to provide the Microsoft Corporation's MS OS/2 operating system with the Department of Defense's Transmission Control Protocol/Internet Protocol (TCP/IP) computer data communication protocols. The motivation for this is to allow users of the MS OS/2 operating system share resources and access to such popular services to as file transfer, electronic mail, and remote program execution that are available on machines and operating systems that have TCP/IP.

TCP/IP [COMER 1988] is a set of computer data communication protocols that define computer to computer communication and conventions for interconnecting networks and routing traffic. The protocols are used extensively on machines that are presently connected to the Defense Advanced Research Projects Agency (DARPA) Internet.

Machines that provide TCP/IP vary from single-user, single-tasking personal computers such as IBM PCs running Microsoft Corporation's MS-DOS operating system to supercomputers such as Cray X-MPs running proprietary operating systems. These machines are interconnected in a multitude of ways that range from slow serial connections such as the modems to high-bandwidth satellite and fiber optic connections [COMER 1988].

In spite of the wide array of machines and operating environments, TCP/IP has provided a means of transparently accessing and utilizing these varied resources. MS OS/2 is a fairly new operating system [LETWIN 1988], and providing it with TCP/IP capability will enhance it's utility and appeal to users.

1.2 Implementation Criteria

This effort has to meet the following criteria:

a) be easy to use for end-users and developers,
b) be non-intrusive to the operating system, and
c) co-exist with networking facilities already provided in the operating system.

Criteria (a) means that applications, such as electronic mail, file transfer, remote procedure

execution, that are written to take advantage of the TCP/IP suite of protocols (Appendix I), should be just as easy to use and be consistent with similar implementations on other operating systems. In this regard, the similarity is provided by using FTP Software, Inc.'s public domain network interface standard called Packet Driver Specification (PDS) Version 1.09 [FTP 1990]. This specification was originally meant for Microsoft Corporation's MS-DOS operating system a single-taking operating system. As the target operating system was a multi-tasking system, modifications were made to the PDS for purposes of this work. These modifications are very minor and are discussed in Chapter 3.

The second criteria (b) implies that the solution should be consistent with the philosophy, design and workings of MS OS/2 [LETWIN 1988]. MS OS/2 was been designed specifically for the Intel 80286 microprocessor. The Intel 80286 allows memory to be accessed in two mutually exclusive modes - real address mode and protected virtual address mode [INTEL 1988].

Further, the Intel 80286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to memory by an application. Figure 1 shows the four levels. The levels

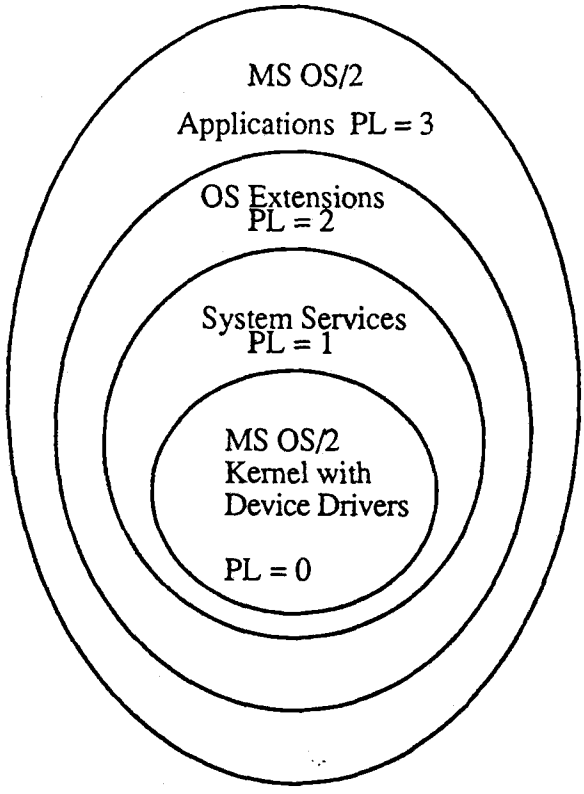

Figure 1: The Privilege Levels of the Intel 80286 Microprocessor and MS OS/2 Kernel Map

are numbered from 0 to 3, with level 0 being the most privileged.

The need to maintain system integrity, especially with reference to memory access rights of applications running on the Intel 80286 microprocessor has, therefore, to be observed religiously. This is assisted greatly by the design of MS OS/2 as far as access to virtual and real memory is concerned [LETWIN 1988].

Criteria (c) is perhaps the most stringent. MS OS/2 currently comes with networking capability built-in. This is called OS/2 LAN Manager (Appendix II). OS/2 LAN Manager provides the user with remote access to other MS OS/2 machines. This allows for sharing of resources such as disks, printers, plotters and modems.

OS/2 LAN Manager, however, is not built upon the TCP/IP suite of protocols, but rather on another standard called NetBIOS [SCHWADERER 1988] and [MICROSOFT 1988]. The services currently provided by OS/2 LAN Manager are very well implemented and highly desirable to have running even when TCP/IP capabilities are added to MS OS/2.

Chapter 2 Alternatives

2.1 Background

After extensive literature searches [KOCHAN 1989], [MICROSOFT 1989], [DUNCAN 1989], [IOACOBUCCI 1988], [NGUYEN 1988] and [WESTWATER 1989], the following alternative implementations were arrived at. They are:

Option A

A standard MS OS/2 device driver that implements the entire Data Link Layer (Layer 2) of the OSI Reference Model (Figure 2). The Data Link Layer provides algorithms for data framing, error, and flow control between two adjacent machines (i.e., these machines are connected by a coaxial cable or a telephone line). The crucial property is that the channel makes the data arrive in exactly the order it was sent [TANENBAUM 1988].

Option B

A protocol device driver that implements portions of IEEE 802.2 Logical Link Layer (LLC) Standard [IEEE 1985]. The LLC forms the upper part of the Layer 2, the Data Link Layer (Figure 3). The protocol device driver will be based entirely on the Network Driver Interface

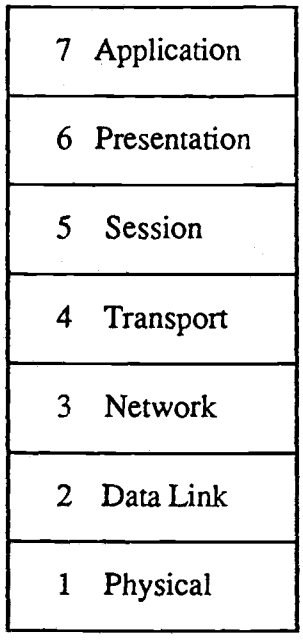

Figure 2: The Seven Layer OSI Reference Model

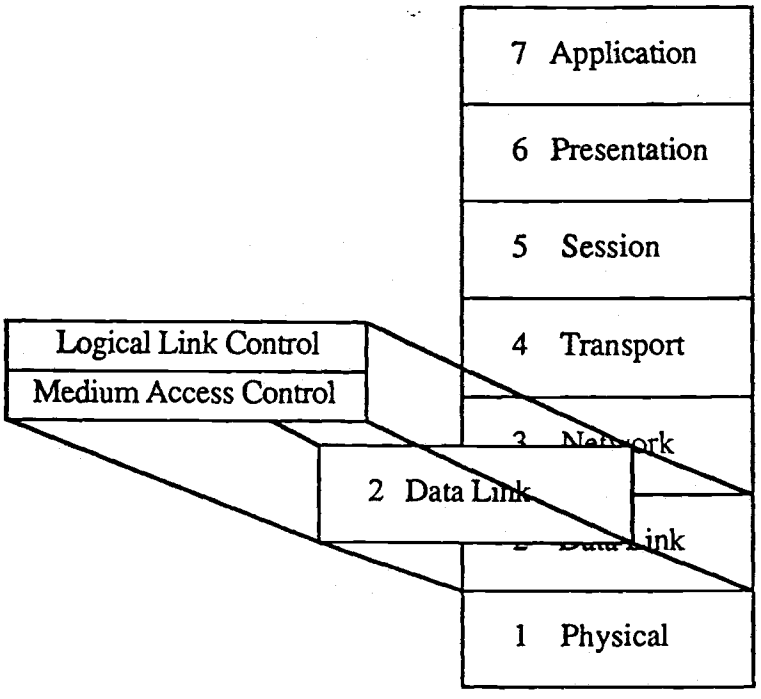

Figure 3: The Layers within the Data Link Layer

Specification (NDIS) Version 2.0.1 produced by 3Com and Microsoft Corporations [3COM 1990].

Option C

A device monitor that attaches itself to an existing NDIS protocol or medium access controller (MAC) device drivers. A device monitor is a program that allows for the user to dynamically monitor any device driver that has been installed in a computer running MS OS/2 (Figure 4). Such device monitors provide a high degree of control of data received and sent by a device driver [LETWIN 1988]. A device monitor is able to add to and delete data that a device driver receives before that device driver passes the data on to any other application. Device monitors are programs that the user activates on a machine that has already been booted up. The monitor therefore is nothing more than a regular MS OS/2 application. Device monitors are MS OS/2 innovations.

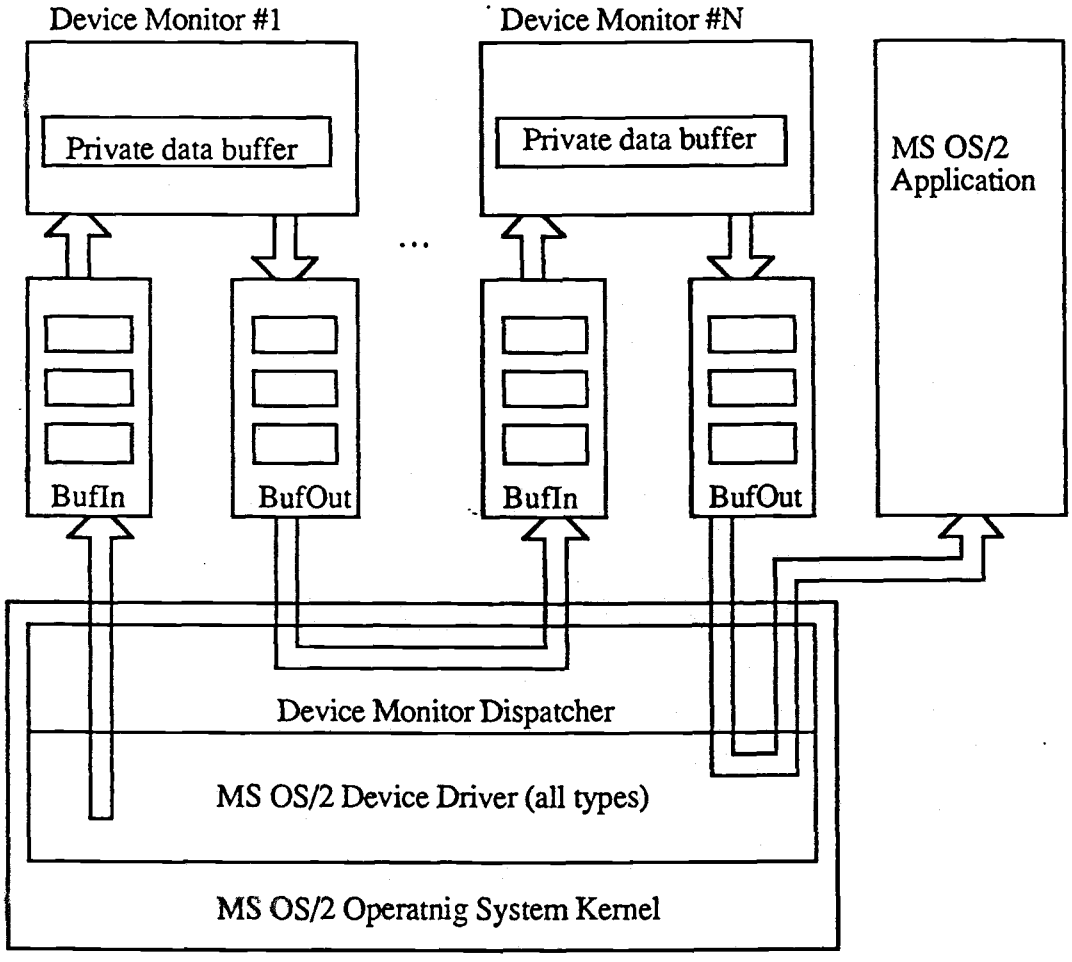

Figure 4: Data stream of the Device Driver moving data through various Device Monitors

Issues relating to the three preceding solutions are discussed below.

2.2 Choosing a Solution

2.2.1 Writing a Standard MS OS/2 Device Driver (option a)

Writing a standard MS OS/2 device driver that implements the entire Data Link Layer has a crucial drawback in that the driver will be specific to a piece of hardware. Such a task is redundant for it has already been undertaken by manufacturers of networking hardware. Also, they write the device drivers to be MAC drivers that adhere to the NDIS. Such an undertaking would be good as an exercise but will mean being tied to one hardware implementation.

2.2.2 Writing a Protocol Device Driver (option b)

The second option is better than the previous option since it is not tied to a specific networking hardware. Further, being conformant to the NDIS, it would mean that the third criteria mentioned in Chapter One, in which OS/2 LAN Manager should remain functional, will be met.

As all OS/2 LAN Manager device drivers are NDIS conformant, this solution would leave OS/2 LAN Manager usable. One possible drawback about writing a packet protocol driver is in the NDIS itself.

The NDIS provides for a single MAC to be able to communicate with multiple protocol device drivers. This is accomplished by a "Vector" inserted at boot time that demultiplexes the data frames amongst the various protocol device drivers expecting data frames from one MAC (Figure 5).

An assumption made here is that the speed of demultiplexing by the vector will not become a performance bottleneck that affects the system throughput. This assumption is justified in that the switching is done on the basis of comparing only one bit in the data stream to decide which protocol device driver to pass the data frame to (Section 4.3 explains this further).

2.2.3 Writing a Device Monitor (option c)

The idea of a device monitor seemed very interesting and was in fact attempted. This solution called for tapping into existing device drivers that receive data

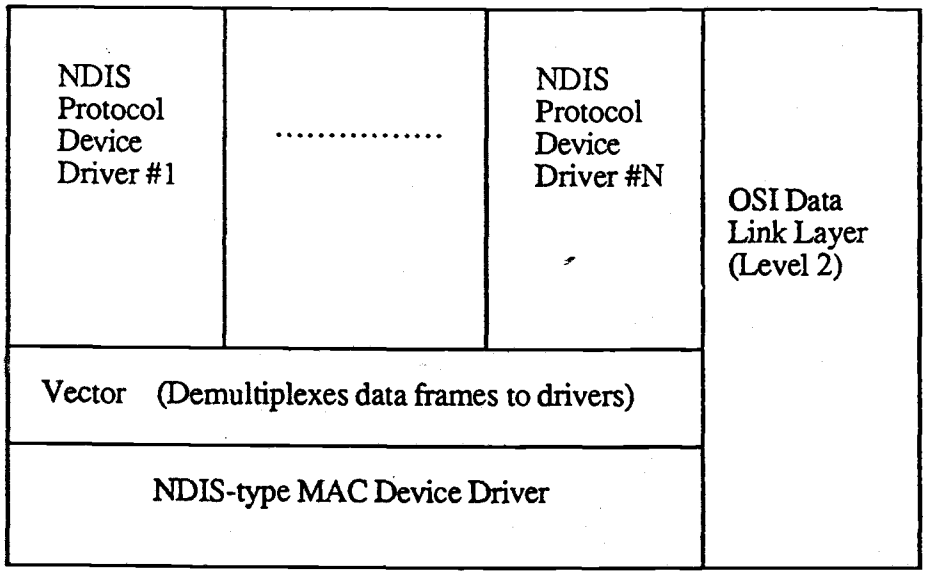

Figure 5: Vector that demultiplexes data frames from one MAC to multiple protocol device drivers

frames from the network. The main advantage of a device monitor is that it can be activated without having to reboot the computer. A device monitor is an application that runs as a standard MS OS/2 application save for the fact that it accesses a device driver.

A device monitor (Figure 4) monitors a data stream of a pre-installed device driver. Any data received by the device driver will either be kept by the device driver or passed on to an appropriate data stream. The device monitor can intercept this data stream and can use/modify/remove/add to the data stream [NGUYEN 1988], [LETWIN 1988].

A fundamental problem encountered in implementing this solution was that the existing device drivers must support device monitor capability. Further, in supporting a device monitor, the device driver will have to predefine a data buffer size that it will make available to a device monitor. There is no guarantee that any of the current OS/2 LAN Manager device drivers provide for device monitors as this is not required in the NDIS.

Also, the speed at which the data winds its way through various device monitors is critical. To achieve maximum performance, it is important that the device

monitor be the first in the chain. Guaranteeing this in practice is not easy. Since device monitors are activated by a command at the MS OS/2 command line, it becomes inelegant to have to dictate a recommended sequence of invoking monitors to end-users.

2.3 Final Choice

From the preceding it is apparent that it would be worthwhile to implement a protocol device driver for it allows independence of underlying networking hardware and permits OS/2 LAN Manager to be functional. Such a driver would provide the following:

a) An NDIS-type interface at the lower level to any pre-existing MAC device driver.

b) An Packet Driver Specification (PDS)-type interface at the upper level that can be accessed by applications that conform to the PDS interface.

c) Provide a device monitoring capability so that system profilers that track system execution, and related other related utilities can be written to monitor performance of the overall design.

Chapter 3 The Solution

3.1 Motivation

As described in Chapter Two Section 2.3, it was decided that a protocol device driver based on the NDIS be written. This protocol device driver is called a packet driver (PD).

Writing a PD immediately provides the ability to keep any existing OS/2 LAN Manager software functional on the same computer. Equally important, the PD allows the reuse of existing TCP/IP software implemented to use the PDS (with minor modifications as noted below).

The PD should be viewed as a piece of software that hides the details required in communicating with the MAC and the networking hardware, from upper level applications. This is in keeping with the computer science precept of modular programming.

3.2 MAC/PD Interface

The PD provides an interface at the lower boundary to the MAC as defined in the NDIS. It is worthwhile to note that this interface definition coincides with the IEEE 802.2 Logical Link Control Standard [IEEE 1985].

The services provided by the PD is a subset of the services defined in the OSI model [TANENBAUM 1988]. Figure 6 shows the relationship between the MAC and the PD, how it relates to TCP/IP protocols and MS OS/2.

The MAC provides services to the PD, and the PD provides services to applications at higher levels (Network Layer) at Service Access Points (SAPs). A SAP is a point at a layer, N, that offers a service which another higher up layer, N+1, can access [ZIMMERMAN 1980]. Each SAP (analogous to a telephone line socket) has an address (analogous to a telephone number to that socket) that uniquely identifies it (Figure 7).

The [IEEE 1985] discusses extensively all the aspects to a 802.2 LLC Standard. The PD only implements a subset of the 802.2 LLC Standard. For purposes of this discussion, only the relevant aspects are mentioned.

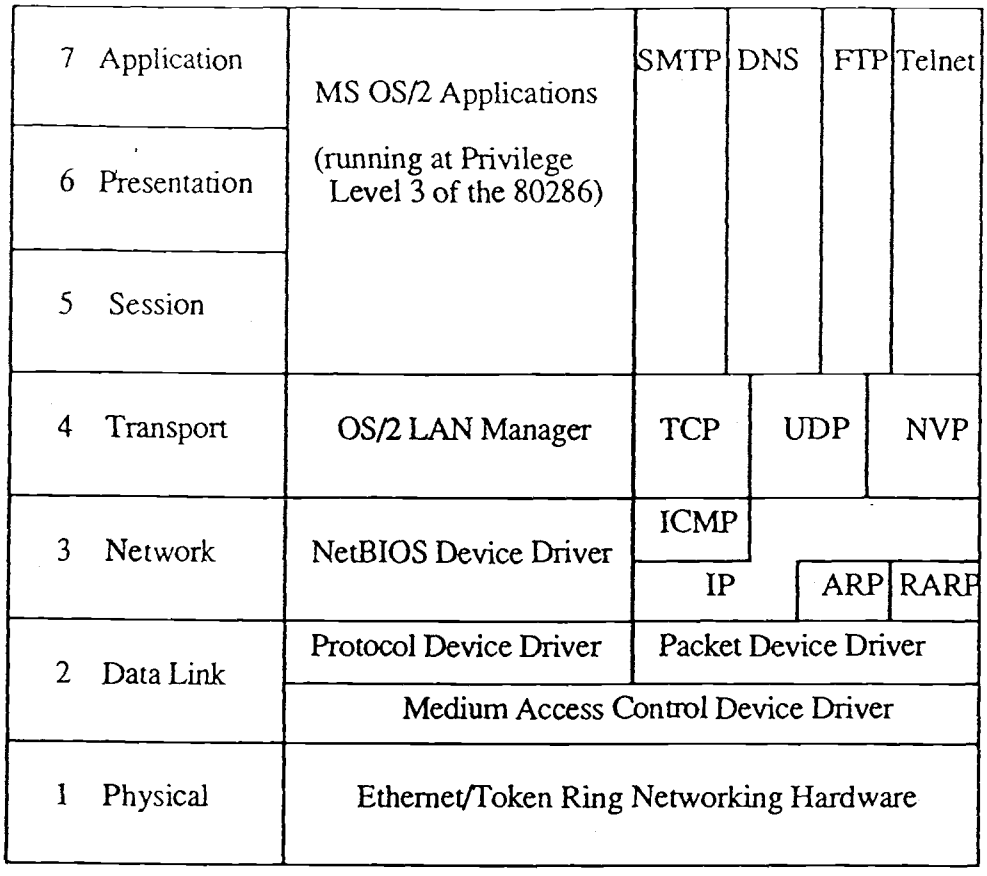

Figure 6: Relationship between the OSI Reference Model, OS/2 LAN Manager, and the TCP/IP Protocol Suite

Legend to Figure 6:

MAC = Medium Access Controller.

NetBIOS = Network Basic Input Output System.

IP = Internet Protocol.

TCP = Transmission Control Protocol - A connection-oriented, reliable, byte-stream protocol.

UDP = User Datagram Protocol - An unacknowledged, transaction-oriented protocol parallel to TCP.

NVP = Network Voice Protocol - Real-time transaction based service for carrying digitized, compressed voice.

SMTP = Simple Mail Transfer Protocol - Provides for sending mail between host machines.

DNS = Domain Name Service - Provides directory service by mapping a machine name to an IP address.

FTP = File Transfer Protocol - Allows file transfer between machines - fairly high speed.

Telnet = Telecommunication Network - Provides virtual terminal service for interactive access by terminal servers to hosts.

ICMP = Internet Control Message Protocol - Used by gateways and hosts on the Internet to appraise other hosts of conditions related to their IP services.

ARP = Address Resolution Protocol - Maps an IP address to its associated Ethernet address.

RARP = Reverse ARP - maps a given Ethernet address to an associated IP address.

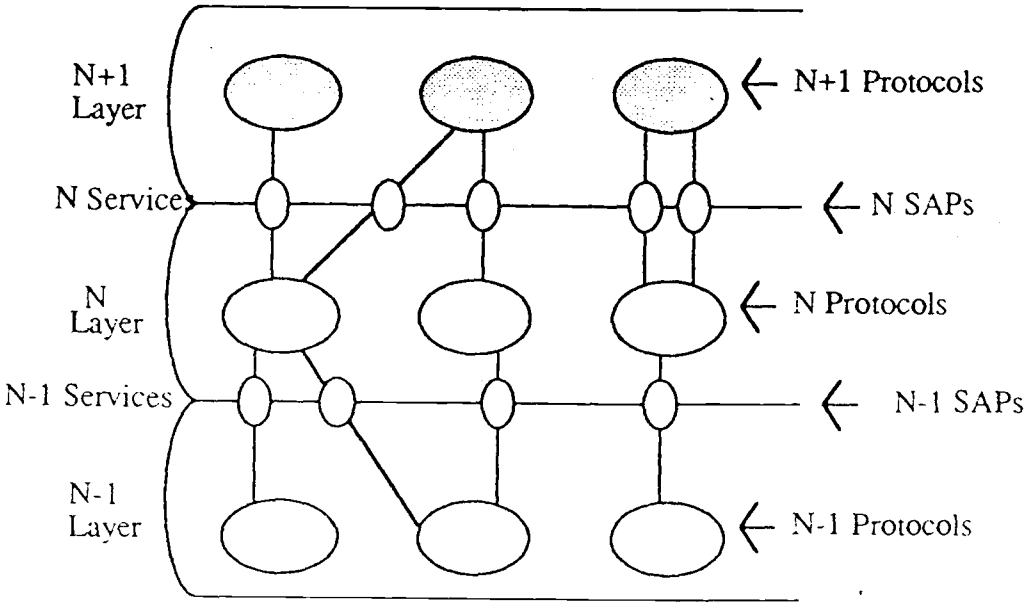

Figure 7: Protocols, Service Access Points (SAPs) and Services [ZIMMERMAN 1980]

The NDIS defines the MAC SAPs (MSAPs) as providing the following to the PD:

MSAP 1:	Pointer to the Common Characteristics Table of the MAC device driver
MSAP 2:	Address to the Request Function
MSAP 3:	Address to the TransmitChain Function
MSAP 4:	Address to the TransferData Function
MSAP 5:	Address to the ReceiveRelease Function
MSAP 6:	Address to the IndicationOn Function
MSAP 7:	Address to the IndicationOff Function

Table 1 - Medium Access Controller Service Access Points

These seven MSAPs are assumed to be available to the PD. The details of the MSAP specifications are not pertinent to this discussion and are not elaborated further. They are defined and explained in the NDIS [3COM 1990].

The flow of data between the MAC and the PD is also described in full detail in the NDIS.

The MSAPs, upon completion of service requests from the PD, calls the PD at specified entry points. A MAC also calls some of these entry points when it receives data from the Physical Layer. These entry points are

defined in Protocol Lower Dispatch Table (PLDT) as per the NDIS and are shown below:

PLDT 1:	Pointer to Common Characteristics Table of the Packet Driver
PLDT 2:	Interface Flag (used by the Vector for data frame dispatch)
PLDT 3:	Address of RequestConfirm Function
PLDT 4:	Address of TransmitConfirm Function
PLDT 5:	Address of ReceiveLookAhead Indication Function
PLDT 6:	Address of IndicationComplete Function
PLDT 7:	Address of ReceiveChain Indication Function
PLDT 8:	Address of StatusIndication Function

Table 2 - Protocol Lower Dispatch Table

It should be reiterated that the PLDTs are not SAPs in that these are points to which the MSAPs return their results after servicing a previous request from the PD or when the MAC received data.

3.3 PD Upper Boundary Interface

The upper boundary of the PD provides Link Service Access Points (LSAPs) that interface with OSI Layer 3, the Network Layer (Figure 2). Applications that access this interface will be running as an regular MS OS/2 application, that is, an Intel 80286 Privilege Level 3 process [INTEL 1988], [LETWIN 1988] (Figure 1).

The LSAPs are defined from the Packet Driver Specification (PDS). The PDS was originally defined for Microsoft Corporation's MS-DOS operating system. Hence, there are some aspects of the PDS that had to be modified to allow it to be useful in the MS OS/2 environment. The modifications are discussed in section 3.6.

The following LSAPs are defined:

LSAP 1:	Address to DriverInfo Function
LSAP 2:	Address to ReceivePkt Function
LSAP 3:	Address to ReleaseType Function
LSAP 4:	Address to SendPkt Function
LSAP 5:	Address to Terminate Function
LSAP 6:	Address to GetAddress Function
LSAP 7:	Address to ResetInterface Function
LSAP 8:	Address to SetRcvMode Function
LSAP 9:	Address to GetRcvMode Function
LSAP 10:	Address to SetMulticastList Function
LSAP 11:	Address to GetMulticastList Function
LSAP 12:	Address to GetStatistics Function
LSAP 13:	Address to SetAddress Function

Table 3 - Link Service Access Points

3.4 Relationship between LSAPs/PLDTs/MSAPs

The correspondance between the LSAPs, PLDTs and MSAPs are summarized in the following table (Table 4). This table groups the services according to their operational similarities.

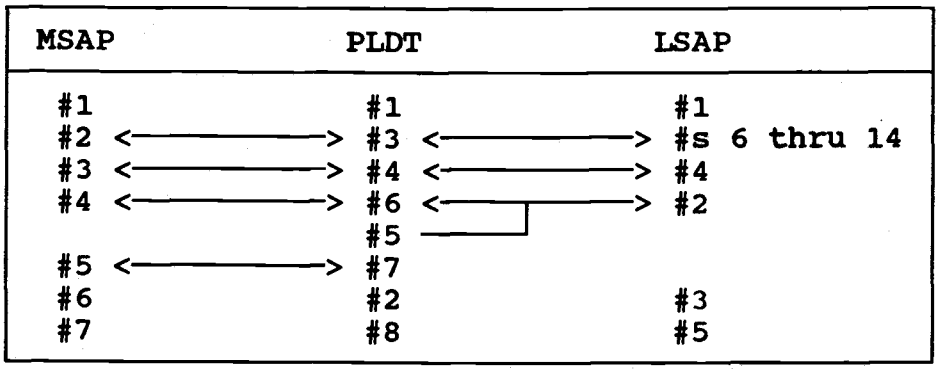

Table 4: Relationship between LSAPs/PLDTs/MSAPs

The table shows associations of the various entry points. Some of these do not have do not have cross relationships for they are administrative function entry points. For example, MSAP #1 and PLDT #1 are pointers to the locations of their respective characteristics tables that store descriptive information of themselves.

The preceding discussion defined the various SAPs at the two sublayers of the Data Link Layer, namely, the Medium Access Layer (MAC) and the Logical Link Control layer (the protocol layer). The software that implements these functions run in the Intel 80286's Privilege Level 0 and is loaded at boot-time.

3.5 Applications Accessing the LSAPs

Having defined the PD's LSAPs, a means of access to these LSAPs has to be provided. As mentioned in Chapter One, a crucial criteria was to be able to provide as easy as possible an access to the network for users and developers of applications written to the PDS. All existing TCP/IP implementations written to the PDS will have to be minimally modified in order to run under MS OS/2.

To use the PD, it has to be loaded into the system at boot time. In MS OS/2 terminology, the file that contains the PD has to be specified in a file called CONFIG.SYS as shown in the following example:

Fragment from CONFIG.SYS:

```
.
.
device=c:\drivers\protman.os2
device=c:\drivers\macwd.sys
device=c:\drivers\packetdr.sys
.
.
```

The first line loads in a program called PROTMAN.OS2 which is a routine that has to be loaded in before any other network device drivers are loaded in. It serves to coordinate the various requests the network drivers may make. Further, this is the program that inserts a vector

should there be more than one protocol driver access on MAC driver.

The second line shows the MAC device driver that is to be loaded. This example represents what was loaded into the development machine (described further in Chapter Four). This is a device driver that is provided by the manufacturer of the networking hardware.

The last line in the fragment above is the name of the file that contains the code for the PD. Successful loading of this file will mean that subsequent operating system requests for opening the PD (as discussed below) will be able to succeed.

Applications can access the PD's LSAPs via a MS OS/2 mechanism called DosDevIOCtl (Appendix IV). As this is the preferred method, the PDS was modified to allow for this scheme.

For example, in order to access LSAP 1, the command syntax will be (in C):

```
.
.
.
Size=0L;          /* a long variable stating
                     the sizeof the new file */
FileAttribute=0; /* file attribute */
OpenFlag=0x0001; /* Open if it exists, if not
                       existing, return an error */
OpenMode=0x0042; /* Set to read-write mode by all */

if ((error=DosOpen( PKTDRVR$, Handle, Action, Size,
                    FileAttribute, OpenFlag,
                    OpenMode, 0h))
     != 0)
   return error;

DosDevIOCtl(Data, List, 01h, DFh, Handle);
.
.
.
```

The DosOpen command is a standard MS OS/2 means of opening a file, in this case the device being opened is called "PKTDRVR$". The DosOpen command is passed the appropriate parameters. DosOpen will return an error number, with which the calling routine can decide if DosOpen succeeded. This routine need be invoked once at the start of the application.

The definitions of the various parameters of DosDevIOCtl command are given in Appendix IV.

3.6 Modifications to Packet Driver Specification (PDS)

The modifications to the PDS are as follows:

a) The software interrupt mechanism in the PDS Section 3, suggests having the interrupt be in the range from 60H to 80H. The exact interrupt number is determined by the application by scanning for a predefined string "PKT DRVR". Such a scheme is fine for MS-DOS. However, in MS OS/2, this is not an elegant solution and hence, the following will be defined.

The software interrupt is predefined at 60H+7FH which gives DFH. The offset 7FH is used because MS OS/2 requires user defined DosDevIOCtls to be in the range 80H to FFH [DUNCAN 1989].

The software interrupt DFH is called the Category Number of the DosDevIOCtl function.

b) The 14 functions defined in the PDS (and provided at the LSAPs), are accessed via the DosDevIOCtl DFH Category call subroutine. The function number is specified in the function number field of the DosDevIOCtl parameter list.

c) The appropriate data structures that each function needs/returns are also defined (Appendix IV).

d) The name of the PDS function AccessType is changed to ReceivePkt to properly reflect its function.

e) The intent of the Terminate Function is modified to mean executing a release of data connection.

Chapter 4 Discussion

4.1 Tools for Software Development

Undertaking this research and development required investment in hardware and software.

4.1.1 Hardware

The following hardware was added to the development machine (a Wyse pc286 personal computer):

a) One forty megabyte hard disk drive.
b) Four megabytes of random access memory.
c) One Western Digital Ethernet (IEEE 802.3) network interface card WD8003E.
d) One standard ASCII terminal connected to the RS232C port (COM1) of the development machine to serve as the debugging terminal.
e) One color monitor and video card.

For developing and testing the code, this setup was attached to the network of the Department of Electrical and Computer Engineering at Oregon State University.

4.1.2 Software

The following software tools were required:

a) MS OS/2 Device Driver Development Kit.
b) MS OS/2 NDIS Network Driver Development Kit.

c) Microsoft C Compiler and Macro Assembler for MS OS/2.

d) Debugging MS OS/2 Kernel.

4.1.3 Miscellaneous Tools

A network analyser that displays the data sent out on a cable was used to track and monitor the data frames and to help in the debugging of the system.

4.2 The Development Cycle

This was the most challenging part of the entire exercise. Writing a device driver is not a trivial task, for it involves very tight interaction (mostly asynchronous) with the host operating system. Maintenance of operating system integrity is very critical. The tools provided in the MS OS/2 device driver development kit helped in this process.

As was alluded to in Chapter Two, the initial thrust was to try and write device monitors. Very quickly, however, it was discovered that such a solution would not work in a general case, for support by NDIS device drivers of device monitors was optional.

The final device driver, the PD (Packet Driver), proved to be a relatively intricate piece of programming. The pseudo-code of the packet driver is enclosed in Appendix III. The packet driver was written entirely in Intel 80286 assembly code and compiled with Microsoft Macro Assembler (MASM) version 5.1.

After the PD was completed, the next task was to modify and recompile an existing MS-DOS TCP/IP implementation to take advantage of the new system. The implementation chosen was "KA9Q Internet Protocol Package" which is used extensively by the amateur radio community and is available in source code, free of royalties from the main author, Phil Karn. The necessary modifications to the code was easily done because the code was modularly written and the portions to change were localized to one module.

4.3 Performance Issues

In the definition of the PLDTs (PLDT 2: The Interface Flag), the PD has to be pre-defined to handle one of the following data frames from the MAC:

a) Flag = 0H: Handles non-LLC data frames

b) Flag = 1H: Handles specific-LSAP LLC data frames

c) Flag = 2H: Handles non-specific-LSAP LLC data frames

The definition of these flags is crucial only when there exists more than one protocol device driver that requires communication with a MAC.

At boot-time, the boot process determines the number of protocols that request access to a specific MAC. The program that does this is called PROTMAN.OS2 (as mentioned in Section 3.5). Should there be more than one per MAC, then the boot process inserts a demultiplexing mechanism called a Vector.

In a simple booting process (one protocol device driver and one MAC), the two device drivers exchange their MSAPs and PLDTs. However, if 2 or more protocols per MAC exist, then this scheme will fail. Here is where the Vector scheme comes in.

The vector will provide the PLDTs to the MAC and receive the MSAPs from the MAC. When a data frame is received from the MAC, the Vector will then determine which of the protocol drivers to send to. The choice is

based upon the value of the Interface Flag, as shown in the following sequence:

a) Protocols handling non-LLC data frames;
b) Protocols handling LLC frames with specific LSAPs;
c) Protocols handling LLC frames with non-specific LSAPs.

In the case of the PD, this flag is set to 0H, meaning that it will handle non-LLC data frames. Although there are defined LSAPs, TCP/IP does not use LLC data frames [TANENBAUM 1988].

Given the preceding, there is a possible performance bottleneck at the Vector. As was mentioned earlier, the assumption can be made that the demultiplexing will be fast for it is being made on the basis of one bit.

Also, it is conceivable that the most likely scenario will be a machine running MS OS/2, OS/2 LAN Manager and the Packet Driver for TCP/IP. Hence, there will only be two protocol drivers (the PD and the OS/2 LAN Manager) accessing one MAC. It is highly unlikely that there will be throughput delays in such a configuration.

4.4 Results

As stated earlier, the PD was written entirely in Intel 80286 Assembly language and compiled using Microsoft Macro Assembler (MASM) Version 5.1. It comprises of header definition files, and small modules that implement each of the LSAP functions. Overall, the program comprises about 900 lines of code.

The PD and the ported TCP/IP package (written in the C language and recompiled using Microsoft C Compiler Version 5.1), currently allows for the successful setting up of a TELNET session to another host.

Chapter 5 Directions for Future Work

5.1 The Next Stage

The current effort concentrated on the development of the PD. This is sufficient in allowing existing MS-DOS TCP/IP implementations to be simply and easily ported over to MS OS/2. This scheme does not take full advantage of the extensive multitasking capabilities of MS OS/2. The next logical step will be to rewrite the TCP/IP code specifically for MS OS/2.

5.2 Extensions to Current Work

The current implementation of the packet driver allows for device monitors to be added. Such device monitors can provide such services as:

a) Data Packet Tracing: Such a capability will give system and network managers a tool to do performance analysis of the network. It will also allow a means of calculating the cost of usage of the network (by way of number of data frames sent and correlating it to the address it was sent to). Further, it can also be used for security monitoring of the data on the network.

b) Computation of data packet transmission and reception times and error statistics. This will be useful for incorporation of dynamic routing algorithms for optimum usage and fine-tuning of the network.

c) Provision of data de/encryption and compression facilities. Such a utility will be very useful in an environment that requires a high degree of security with both incoming and outgoing data packets. The data packets can be de/encrypted using well-know and highly secure algorithms such as the Data Encryption Standard (DES).

Chapter 6 Conclusion

The entire undertaking was to prove one simple point: that it is possible to provide MS OS/2 with TCP/IP capabilities. Giving MS OS/2 TCP/IP capabilities means that the end user now can have access to a wide array of services and resources on a multitude of computing platforms. This was accomplished with the development of a protocol device driver that amalgamated two different interface strategies.

Also, as a final result, now TCP/IP services that were available to MS-DOS users are functional in MS OS/2 - with exactly the same user interface.

The learning experience offered in this exercise was very rewarding. This was because of the need to gain familiarity with a new operating system, to get acquainted with the tools involved and to provide a stable and viable platform to build upon for the future.

Bibliography

[3COM 1990] 3Com Corporation and Microsoft Corporation, Microsoft/3Com LAN Manager Network Driver Interface Specification Version 2.0.1 Published February 26, 1990

[COMER 1988] Douglas Comer, Internetworking with TCP/IP: Principles, protocols, and architecture, Prentice-Hall, New Jersey. ISBN 0-13-470154-2

[DAVIDSON 1988] John Davidson, Introduction to TCP/IP, Springer-Verlag, New York. ISBN 0-387-96651-X

[DUNCAN 1989] Ray Duncan, Advanced OS/2 Programming, Microsoft Press, Washington. ISBN 1-55615-045-8

[FTP 1990] FTP Software Inc., PC/TCP Packet Driver Specification version 1.09, FTP Software Inc., Massachusetts. Document is in public domain.

[IACOBUCCI 1988] Ed Iacobucci, OS/2 Programmer's Guide, Osborne McGraw-Hill, California. ISBN 0-07-881300-X

[IEEE 1984] IEEE, IEEE Standards for Local Area Networks: Logical Link Control 802.2, IEEE Inc, New York. ISBN 0-471-82748-7

[INTEL 1988] Intel Corporation, Microprocessor and Peripheral Handbook Volume I Microprocessor, Intel Corporation, California. ISBN 1-55512-073-3

[KOCHAN 1989] Stephen G. Kochan and Patrick H. Wood, Unix Networking, Hayden Books, Indiana. ISBN 0-672-48440-4

[LETWIN 1988] Gordon Letwin, Inside OS/2, Microsoft Press, Washington. ISBN 1-55615-117-9

[MICROSOFT 1988] Microsoft Corporation, Programming Interface for the OS/2 LAN Manager, Microsoft Corporation, Washington. Microsoft Part Number 01396

[MICROSOFT 1989] Microsoft Corporation, Network Device Driver Kit, Microsoft Corporation, Washington. Microsoft Document Number: SY0829-100-000-0389

[NGUYEN 1988] Thuyen Nguyen & Robert Moskal, Advanced Programmer's Guide to OS/2, Simon & Schuster, Inc, New York. ISBN 0-13-642935-1

[SCHWADERER 1988] W. David Schwaderer, C Programmer's Guide to NetBIOS, Howard W. Sams & Company, Indiana. ISBN 0-672-22638-3

[TANENBAUM 1988] Andrew S. Tanenbaum, Computer Networks Second Edition, Prentice-Hall, New Jersey. ISBN 0-13-162959-X

[WESTWATER 1989] Raymond Westwater, Writing OS/2 Device Drivers, Addison-Wesley Publishing Co. Inc, Massachusetts. ISBN 0-201-52234-9

[ZIMMERMAN 1980] Hubert Zimmerman, OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on Communications, Vol. COM-28 No. 4, April 1980 pages 425-432, IEEE Inc, New York.

Appendices

Appendix I

A Brief Overview of TCP/IP and OS/2 LAN Manager

The OSI Reference Model (Figure 1) shows the various layers in a networking system and describes the functions they provide. This model actually specifies a communication service [DAVIDSON 1988]. Each layer provides a service to the next layer above through SAPs.

Figure 6 shows the various components that is collectively called the TCP/IP protocols suite. IP provides services to the Transport Layer from the Network Layer. IP expects services from the Data Link Layer. Similarly, TCP provides services to the Session Layer from the Transport Layer.

The two protocols TCP and IP co-exist with other protocols such as ICMP, ARP, RARP, UDP and NVP. Despite these other protocols, the name TCP/IP is used to refer to all of them, not just TCP and IP.

These protocols are augmented by other services that run in the Session and higher layers. Popular services are telnet, rlogin (remote login), ftp and, smtp.

The critical issue here is that TCP/IP now has been implemented on numerous computing platforms used for both research, academic and commercial purposes.

From Figure 6, it is apparent how the communication stack of TCP/IP compares with that of OS/2 LAN Manager. The current interface of OS/2 LAN Manager to the network is via NetBIOS [SCHWADERER 1988]. In an OS/2 LAN Manager installation, therefore the protocol driver interacts with a NetBIOS driver. Conceptually, the NetBIOS driver resides at the Network Layer like IP.

Both IP and NetBIOS provide totally different addressing schemes. However, there is now a standard that allows for the encapsulation of IP datagrams to be sent over a NetBIOS network (called Request For Comment (RFC) 1001, RFC 1002 and RFC 1088). These RFCs are attainable from NIC.DDN.MIL (IP address 26.0.0.76).

Appendix II

A Brief Overview of MS OS/2 and Intel 80286 Privilege Levels

MS OS/2 is an operating system written specifically for the Intel 80286 microprocessor. The Intel 80286 provides 2 modes of operation - protected and real modes.

When the 80286 is running in protected mode, MS OS/2 is able to access more hardware registers and segment registers that are otherwise unavailable in the real mode - the mode in which MS-DOS runs.

Access to these extra registers allows for the design of an efficient and secure multitasking operating system. It provides the following:

a) Protection of the code and data of the OS from applications;
b) Protection of system resources from being interfered with by each application;
c) Allowing for an address space per application to up to 1 gigabyte through the use of a virtual memory system;
d) Providing a fast and efficient task-switching mechanism.

The protection provided to the system is made available via an Intel 80286 concept called privilege levels (or rings). Under this scheme, any code running under MS OS/2 is automatically assigned a privilege level. The Intel 80286 provides support for 4 hardware recognized privilege levels arranged hierarchically from 0 being the highest and 3 the lowest.

Hence, code and data at a higher privilege level is inaccessible to code at a lower level, but the reverse is true. Therefore, by having MS OS/2 run at the highest level, it effectively is protected from other programs.

Of the 4 levels, MS OS/2 only uses 3 (0, 2 and, 3). Level or ring 0 is reserved for MS OS/2's kernel. Level 2 for programs that need input/output privileges and, level 3 for application programs. Hence, device drivers reside at the ring 0 level as they become part of the MS OS/2 kernel after boot time.

Appendix III

Packet Driver Pseudocode

The following is the pseudocode of the packet driver. The packet driver was written in Intel 80286 assembly language using Microsoft Macro Assembler MASM Version 5.1.

```
define constants, error and progress messages
define DevHlp function numbers
define structures for packet driver
define standard device driver header
define tables for NDIS - Lower Dispatch Table
       and Common Characteristics Table
define driver command table

Standard Device Driver Strategy Routine
     Save pointer to data packet for OS kernel
     Check for valid function (LSAP);
     and call function (LSAP)
     else return error code to calling application;

DriverInfo LSAP
     Return driver characteristics to calling
     application;

ReceivePkt LSAP
     Activate data receiver of user application;
     If unable to activate, discard data;
     Transfer data to data receiver;

ReleaseType LSAP
     Close any opened handles to current
     application;
     If non-existent then return error;

SendPkt LSAP
     Accept data frame from application;
     Invoke MSAP TransmitChain and pass data frame
     and unique handle;

Terminate LSAP
     Close data connection if it exists
     else return error;

GetAddress LSAP
     Return error to calling routine - no match in
     NDIS-type MAC;
```

```
ResetInterface LSAP
     Reset interface associated with handle by
     calling MSAP Request Function ResetMAC;
     Pass unique handle to MSAP and await it's
     return;

SetRcvMode LSAP
     Set receiver mode of network device by
     calling MSAP Request Function SetPacketFilter;
     Pass unique handle to MSAP and await for it
     when MSAP returns;

GetRcvMode LSAP
     Return error to calling routine - no match in
     NDIS-type MAC;

SetMulticastList LSAP
     Call MSAP Request Function AddMulticastAddress
     with addresses to be added as provided by
     calling application;
     Pass unique handle to MSAP and await for it
     when MSAP returns;

GetMulticastList LSAP
     Return error to calling routine - no match in
     NDIS-type MAC;

GetStatistics LSAP
     Call MSAP Request Function UpdateStatistics;
     Pass unique handle to MSAP and await for it
     when MSAP returns, return updated data to
     calling application;

SetAddress LSAP
     Call MSAP Request Function SetFunctionalAddress
     with received address;
     Pass unique handle to MSAP and await return;

RequestConfirm PLDT
     MAC invokes this routine;
     Check for handle returned by MAC if Requests to
     MSAP Request Function was initiated.

TransmitConfirm PLDT
     MAC invokes this routine after successfully
     sending packet of previous SendPkt call;
     Check handle returned by MAC for consistency;

ReceiveLookAhead PLDT
     MAC invokes this routine upon receipt of
     new data on network;
     Accept all data by calling MSAP TransferData;
```

```
IndicationComplete PLDT
     Acknowledgement from MAC that data accepted
     by Protocol can now be processed;

ReceiveChain PLDT
     MAC invokes this routine upon receipt of
     new data on network;
     Accept all data by calling MSAP ReceiveRelease;

StatusIndication PLDT
     MAC calls that are non-data reception
     initiated;
     Accept data from MAC showing status of
     network hardware;

SystemRequest Initialization-time Function
     Accept call from PROTMAN.OS2 to perform
     binding with specified MAC driver;
     Call MAC's Bind Routine via MAC's
     Common Characteristics Table;
     Return to PROTMAN.OS2 binding request
     status;

Packet Driver Initialization Function
     Invoked at boot-time;
     Save addresses to DevHlp, address of
     data segment of module;
     Perform DosOpen on PROTMAN.OS2;
     If DosOpen successful, call PROTMAN.OS2
     to RegisterModule and await SystemRequest
     call from PROTMAN.OS2
```

Appendix IV

DosDevIOCtl Specification

The Packet Driver will be accessed with the DosDevIOCtl function call as defined below:

```
DosDevIOCtl(Data, ParmList, Function, Category,
            DevHandle)

    char far *Data;          /* pointer to the data
                                block to be returned */

    char far *ParmList;      /* pointer to the parameter
                                list */

    unsigned Function;       /* function number */

    unsigned Category;       /* category number */

    unsigned DevHandle;      /* device handle */
```

The first parameter, Data, is a pointer to a memory block where the device driver returns the data. The second, ParmList, is a pointer to a memory block containing the input parameters. The third parameter, Function, is an unsigned or 2 byte variable the specifies the function number of the IOCTL function. The function number is the number of the Link Service Access Point (LSAP) as defined in Table 3. The fourth parameter, Category, is another unsigned variable that specifies the category of the IOCTL function, which in the case of the packet driver is DFH. The last parameter is a number that is supplied by the caller. This number was obtained

by the caller by issuing a DosOpen call before doing a DosDevIOCtl function call.

An application wanting to use the packet driver will have to perform a DosOpen on the device called PKTDRVR$. If this succeeds, then the packet driver is currently installed and accessible and the application can issue DosDevIOCtl function calls.

The data structures for the various functions that can be accessed via the DosDevIOCtl function call are defined below:

Function 1: driver_info

```
struct driver_info {
     int error;      /* same as PDS */
     int version;    /* BX in PDS */
     int class;      /* CH in PDS */
     int type;       /* DX in PDS */
     int number;     /* CL in PDS */
     int basic;      /* AL in PDS
                        basic=01, extended=02
                        not installed=FF */
}
```

Function 2: access_type

```
struct access_type {
     int if_class;      /* AL; Interface class */
     int if_type;       /* BX; Interface type */
     int if_number;     /* DL; interface number */
     char far * type;   /* DS:SI; access type */
     unsigned typelen; /* CX; access type length */
     int (far *receiver) ();  /* location of
                                 receiver routine
                                 of application */
     int error;               /* same as PDS */
}
```

```
Function 3: release_type

struct release_type {
     int handle;     /* error if handle has error */
}


Function 4: send_pkt

struct send_pkt {
     char far * buffer; /* DS:SI; frame to send */
     unsigned length;   /* CS; length of frame  */
     int error;         /* same as PDS */
}


Function 5: terminate

struct terminate {
     int handle;     /* BX; unique handle */
     int error;      /* same as PDS */
}


Function 6: get_address

struct get_address {
     int handle;          /* BX; unique handle */
     char far * buffer;   /* ES:DI; address */
     unsigned length;     /* CX; address length */
     int error;           /* same as PDS */
}


Function 7: reset_interface

struct reset_interface {
     int handle;     /* BX; unique handle */
     int error;      /* same as PDS */
}


Function 8: set_rcv_mode

struct set_rcv_mode {
     int handle;     /* BX; unique handle */
     int mode;       /* CX; mode to set to */
     int error;      /* same as PDS */
}
```

Function 9: get_rcv_mode

```
/* structure defined for consistency. Packet driver
 ignores data and returns "invalid" error message */

struct get_rcv_mode {
     int handle;     /* BX; unique handle */
     int mode;       /* CX; mode currently in */
     int error;      /* same as PDS */
}
```

Function 10: set_multicast_list

```
struct set_multicast_list {
     char far * addrlist; /* ES:DI; address list */
     int length;       /* CX; address list length */
     int error;        /* same as PDS */
}
```

Function 11: get_multicast_list

```
/* structure defined for consistency. Packet driver
 ignores data and returns "invalid" error message */

struct get_multicast_list {
     char far *addrlist; /* ES:DI; address list */
     int length;      /* CX; address list length */
     int error;       /* same as PDS */
}
```

Function 12: get_statistics

```
struct get_statistics {
     int handle;          /* BX; unique handle */
     char far * stats;    /* DS:SI; same as PDS */
}
```

Function 13: set_address

```
struct set_address {
     char far *addr;      /* ES:DI; address list */
     int length;          /* CX; length of list */
     int error;           /* same as PDS */
}
```