DESIGN OF A NETWORKED CD-ROM FOR MULTIMEDIA APPLICATIONS

by

Steven Michael Levis

Submitted to the

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

in partial fulfillment of the requirements

for the degrees of

BACHELOR OF SCIENCE

and

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1995

Author ....
Department of Electrical Engineering and Computer Science
May 15, 1995

Certified by ....
Stephen K. Burns, Technical Dir Of Biomedical Engineering Ctr
Harvard-MIT Health Science
Thesis Supervisor

Certified by ....
David L. Waring, Bellcore
Thesis Supervisor

Accepted by ....
F. R. Morgenthaler
Chair, Department Committee on Graduate Students

Design of a Networked CD-ROM for Multimedia Applications

by

Steven Levis

Submitted to the Department of Electrical Engineering and Computer Science on May 7, 1995, in partial fulfillment of the requirements for the degrees of Bachelor of Science and Master of Science in Electrical Engineering.

Abstract

Allowing users of multimedia applications access to a large library of such items via the networking of PC CD-ROM drives (Net-CD) is investigated in this paper. A model for multimedia systems and an architecture for the implementation of this model are discussed. A simple Net-CD system is designed, constructed, and tested to determine the feasibility of networking CD-ROM drives on a PC.

Thesis Supervisor: Stephen Burns
Title: Technical Director Of Biomedical Engineering Center

Acknowledgements

This project was accomplished during the author's six month cooperative study program with Bell Communications Research (Bellcore). The author would like to thank Bellcore for supporting this project, as well as Alex Gelman and Dave Waring for providing inspiration and opportunity throughout the course of the research.

Table of Contents

1 Introduction 11
2 Multimedia Systems 13
  2.1 A Model for Multimedia Systems 13
  2.2 Video Dial Tone Architecture 14
3 Design of Networked CD-ROM 17
  3.1 Networked CD-ROM on VDT 17
  3.2 Net-CD Software Requirements 18
4 Design of Net-CD Software 23
  4.1 Operation of MSCDEX 23
  4.2 Design of LOC-CD.SYS 24
  4.3 Communication Protocol 24
  4.4 Design of REM-CD.EXE 26
5 Results 27
6 Conclusion 29
Appendix A LOC-CD.ASM 31
Appendix B REM-CD.ASM 55

List of Figures

Figure 2.1: Multimedia System Model 14
Figure 2.2: Video Dial Tone System Architecture 16
Figure 3.1: VDT Implementation of a Networked PC CD-ROM 19
Figure 3.2: PC CD-ROM Software Interaction 20
Figure 3.3: Net-CD Software Interaction 20

List of Tables

Table 4.1: Driver Commands Supported by Net-CD 23
Table 4.2: Protocol for Net-CD Data Transmission 25

Chapter 1

Introduction

Multimedia is the term for applications which appeal to more than one of the senses. These applications require a high-bandwidth information flow to the user. There is a wide variety of such applications available for PCs on CD-ROM. The amount of information available on one CD-ROM is hundreds of times greater than the capacity of standard 3-1/2 inch magnetic diskettes. Information in this large quantity lends itself to use in many types of applications such as video games, encyclopedias, video shopping, and the like.It would be an advantage to the user of multimedia CD-ROM software if his or her PC was networked to a large quantity of CD-ROM type applications. This paper discusses solutions to the networking of a PC CD-ROM and the implementation of one such system.

Chapter 2

Multimedia Systems

2.1 A Model for Multimedia Systems

Figure 2.1 diagrams a model of current multimedia systems. This model shows the general asymmetry of bandwidth between the user and the multimedia information source. Although one can envision a system in which the user input was of a high bandwidth, such as an application with video input, today's multimedia applications have low bandwidth user input such as a mouse, keyboard or remote control. The high bandwidth output of the system is required by applications which output video or audio. This bandwidth asymmetry plays a key role in the system architecture that will be discussed.

The system model in Figure 2.1 is separated into the user interface and the multimedia information source. One example of a user interface is a personal computer, capable of playing audio and video, through which multimedia applications a used. Another is the combination of a television (for audio and video output) and a set-top (for user interface) which would be used for interactive multimedia applications, including video on demand.

By separating the model in this manner the user interface can be made almost independent of the content of the multimedia experience. The exception to this is the necessity of a standard in terms of the data communication and the types of input and output media. In this way, the user can have access to a large multimedia library through a constant interface.

The multimedia information source can be represented in many ways. The source can be a video library, a source of processing power for applications such as central computing and video games, a CD-ROM, or any source of multimedia applications. The information source can also be implemented as a network of many different information providers (IP) which will provide the user with a variety of multimedia and will also allow for competition between IP's.

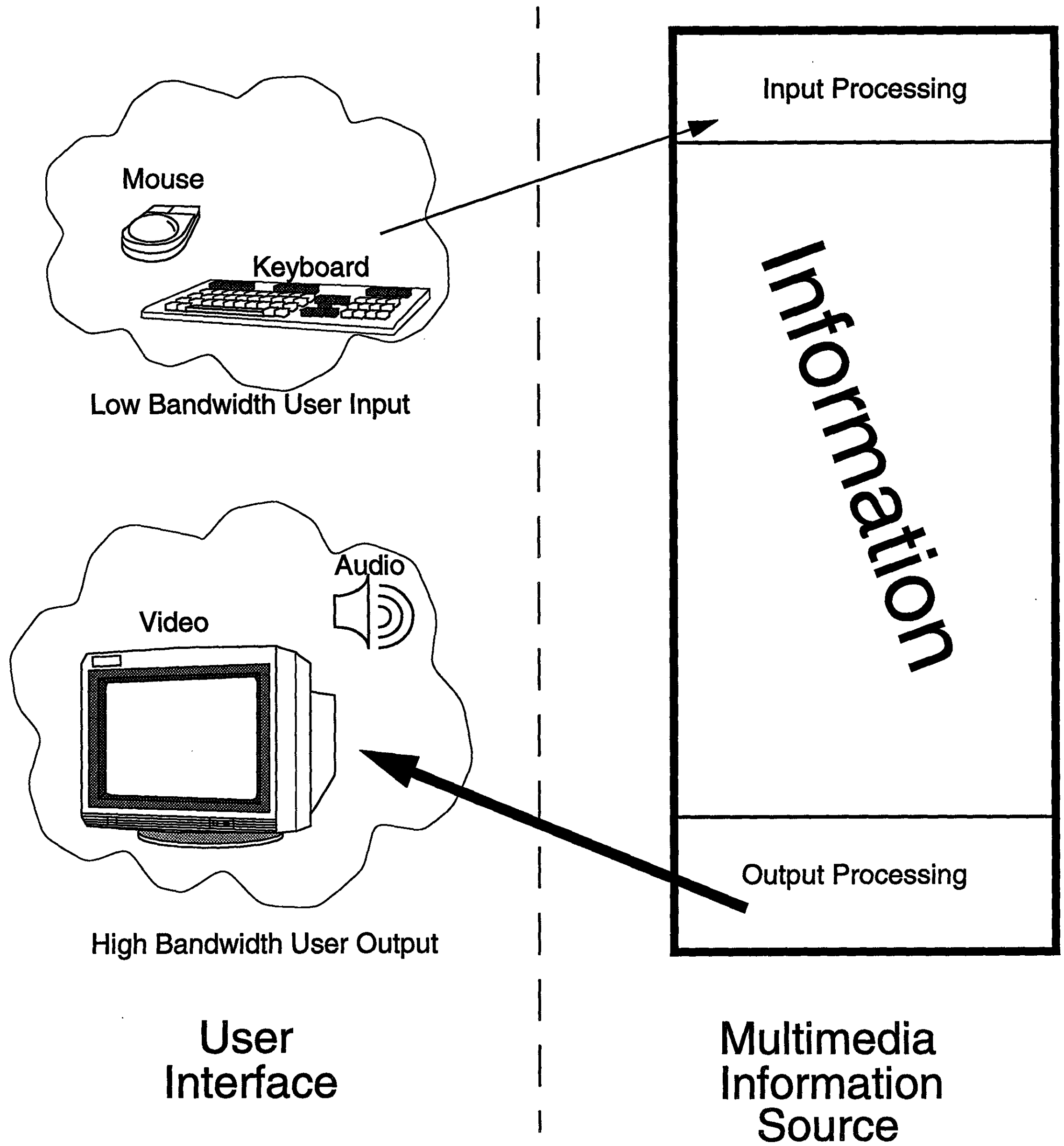

Figure 2.1: Multimedia System Model

2.2 Video Dial Tone Architecture

The Video Dial Tone (VDT) multimedia system architecture takes advantage of, (1) a network already in existence and (2) the asymmetric nature of contemporary multimedia.

The telephone network is used as the channel between the user interface and information providers. By the use of this network, ordinary telephone subscribers will already have access to a data communications channel. The disadvantage of this system is that the user is connected to the network via twisted pair copper wire. The twisted pair, in its current configuration, has quite a low bandwidth relative to the requirements of multimedia output.

The solution to this problem comes in the form of a technology known as the Asymmetric Digital Subscriber Line (ADSL). ADSL allows the normally low bandwidth twisted pair to act as a data communications channel with low bandwidth in one direction (9600 bits/sec) and a high bandwidth in the opposite direction (1.522 Megabits/sec). Because of the inherent asymmetry of today's multimedia applications, ADSL allows the telephone network to be used as a multimedia information source with a small modification to the twisted pair terminations at the subscriber and at the central office (CO).

A diagram of the VDT system architecture is shown in Figure 2.2. The ADSL twisted pair line links the user interface hardware, or Customer Premises Equipment (CPE), with a dedicated VDT line card in the central office. Through the CO, the line card is connected to other CO's and the information providers through a Synchronous Optical NETwork (SONET) using the Asynchronous Transfer Mode (ATM) protocol. SONET is a high bandwidth fiber-optic network which links distant portions of the telephone network.

The VDT line card can be considered the dividing line between the user interface and the multimedia information source in the model shown in Figure 2.1. This line card contains a processor capable of running scripts downloaded into it, as well as a buffer used as a temporary storage area for multimedia information. The line card is capable of executing protocols necessary for communication with both the user interface and the IP. Because of the much higher bandwidth of SONET relative to the ADSL, multimedia information can be sent from the IP in large bursts, buffered in the line card and then sent over the twisted pair at the proper speed. This allows SONET to be time-multiplexed on a per-user basis.

The VDT architecture allows a currently existing network to implement a multimedia information source with little modification to the existing line. This will give users an easy connection to a large library of multimedia titles.

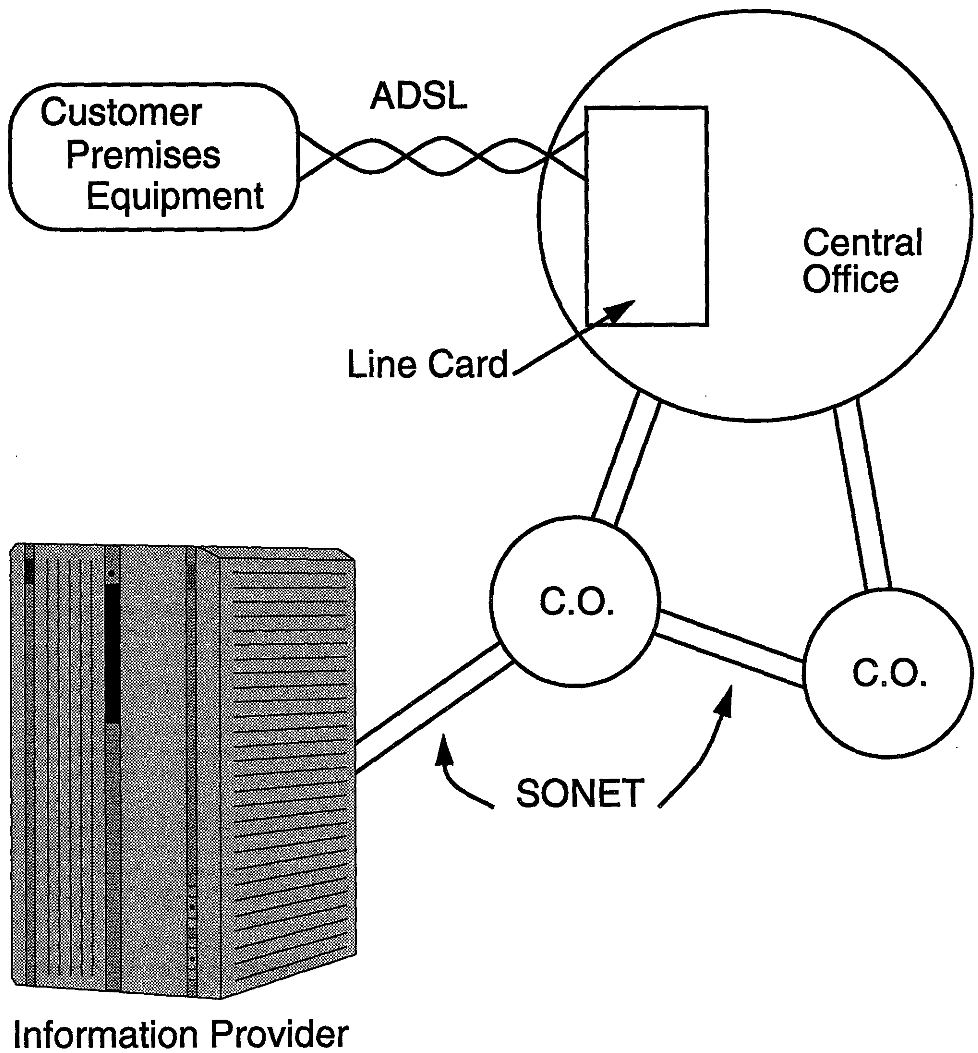

Figure 2.2: Video Dial Tone System Architecture

Chapter 3

Design of Networked CD-ROM

3.1 Networked CD-ROM on VDT

One application that would be well suited for VDT would be that of supplying the user with a large library of PC CD-ROM titles. A diagram for the implementation of such a system is shown in Figure 3.1. The CPE for the networked CD-ROM (Net-CD) would consist of the customer's personal computer and an interface peripheral. The interface would create a data link between the PC and line card, through the SONET and eventually to the IP.

There are many ways to implement the CD-ROM titles in the IP. The IP could be simply another PC with a CD-ROM drive and an interface to the network. A more practical solution might consist of a server which has fast access to many CD-ROM titles. The line card might buffer and cache data in order to improve performance.

The user load on the IP is limited by the bandwidth of its connection to the network. One 100 Megabit/sec line, for instance, could service up to 65 users. In order to increase its capacity, the IP could purchase more lines to increase its bandwidth. This would, however, also require equipment which could handle the added load. The load on the IP is limited by its own resources and by the market.

The architecture of the VDT network has already been developed by engineers at Bell Communications Research (Bellcore). In order to implement Net-CD, software drivers and VDT interface hardware must designed for the CPE. The following section describes the design of the required software for the customer's PC.

3.2 Net-CD Software Requirements

The design of the necessary software to implement Net-CD required research into MS-DOS's implementation of CD-ROM drives in general. Unfortunately, DOS cannot handle CD-ROM drives in the same manner as it does for regular disk drives, due to the large size of data files that is allowed on CD-ROM drives.

DOS requires CD-ROM drives to be supplied with a software driver which will allow communication between the drive and Microsoft's CD-ROM Extensions (MSCDEX). MSCDEX masks the workings of the CD-ROM drive from DOS in a manner that makes the CD appear as if it were a network drive. This is done because DOS allows an exception for network drives in terms of maximum file size.

Figure 3.2 shows the links between DOS, MSCDEX, DRIVER.SYS (the CD-ROM drive specific device driver), and the CD-ROM drive. Both of the first two sections, DOS and MSCDEX, are independent of the type of CD-ROM drive in the system. The final two sections are device dependent and therefore cannot be implemented in the PC if the CPE is to be made as general as possible. This division into device independent and dependant sections is illustrated by a vertical line in Figure 3.2.

The feasibility of implementing the Net-CD on a PC can be shown by writing and testing software which allows us to replace the vertical line in Figure 3.2 with a division in hardware, such as the ADSL line in the VDT architecture. This is shown in Figure 3.3.

In this configuration, MSCDEX communicates with LOC-CD.SYS, which, to MSCDEX, looks like a CD-ROM device driver. LOC-CD.SYS is linked to REM-CD.EXE in the remote system via a communications channel. REM-CD.EXE, on the other hand, looks like MSCDEX to the device dependant DRIVER.SYS.

The communications channel would be replaced by the telephone network between the CPE and the server in the actual implementation of Net-CD using VDT. However, for this experiment, it is a 115.2kbit/sec serial channel. Although this is roughly 12% of the speed of system limiting ADSL speed of 1.522 Megabits/sec, it will not effect the feasibility of a PC Net-CD, as long as a robust CPE interface to the ADSL line can be created.

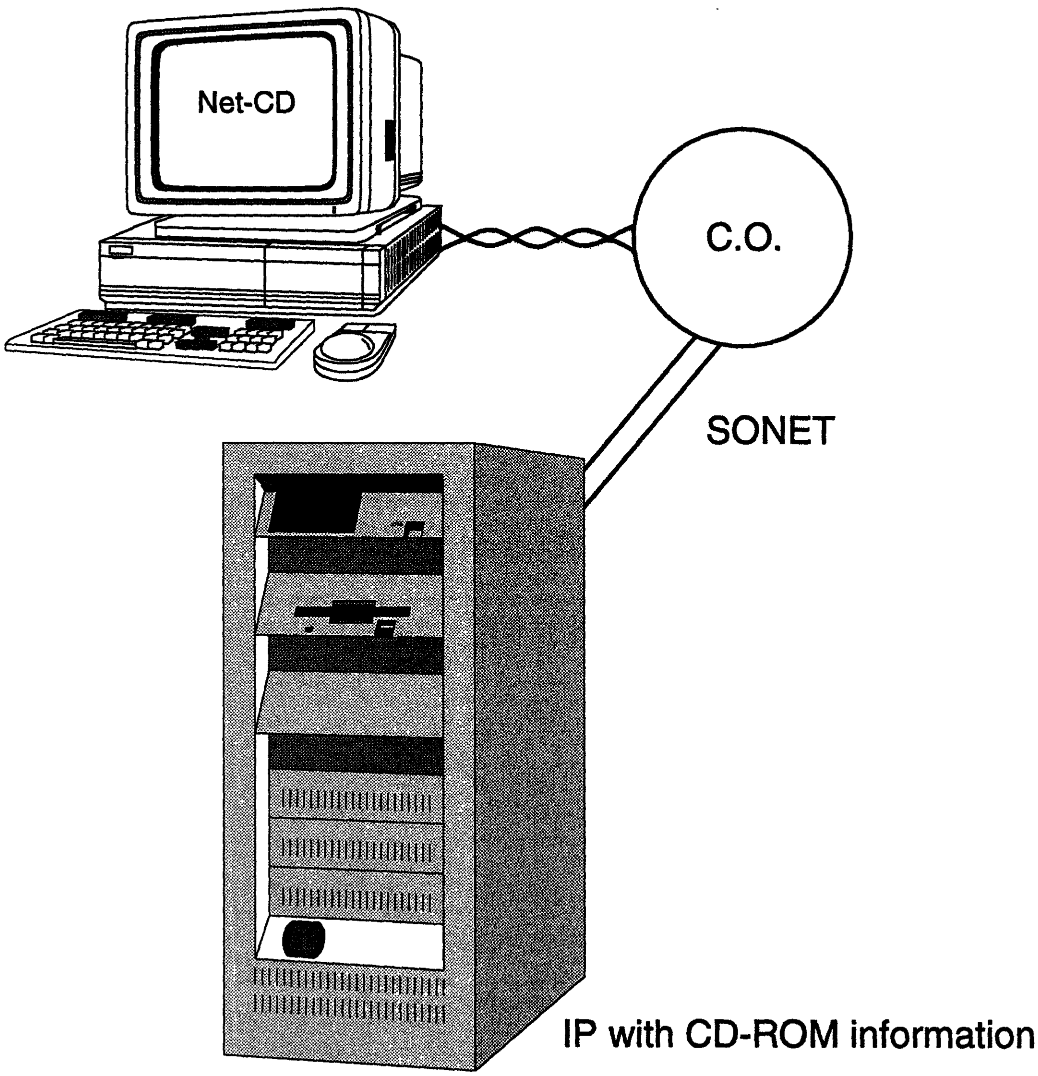

Figure 3.1: VDT Implementation of a Networked PC CD-ROM

The LOC-CD.SYS software driver will receive commands from MSCDEX, and group them into one of two categories. The first consists of all commands which do not require any interaction with the CD-ROM device. All requests of this type will be answered locally. The remaining commands fall into the second category. These requests are placed into a packet for shipment to the remote PC's REM-CD.EXE.

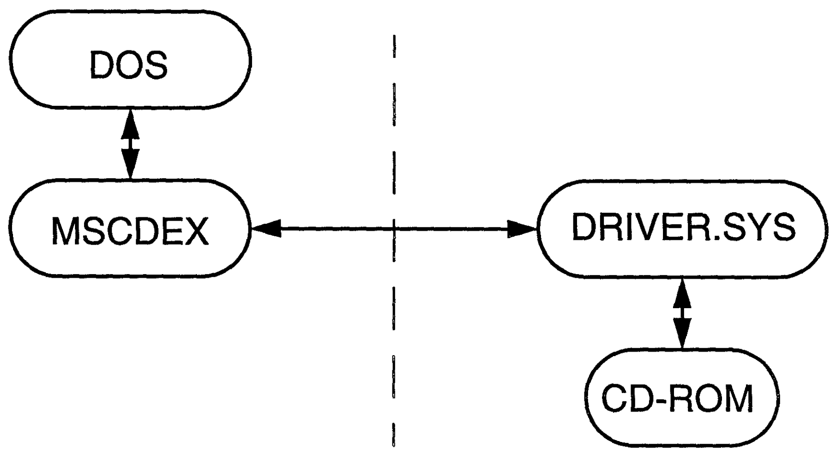

Figure 3.2: PC CD-ROM Software Interaction

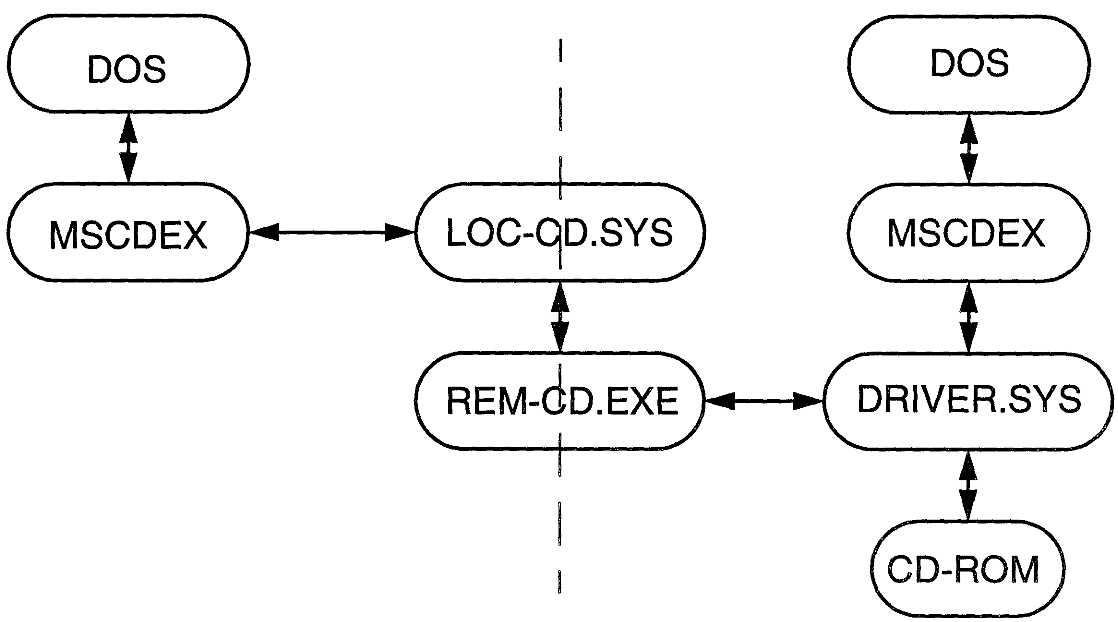

Figure 3.3: Net-CD Software Interaction

The REM-CD.EXE receives commands from the LOC-CD.SYS and passes them to DRIVER.SYS as if they were coming directly from MSCDEX. When the driver returns the request, REM-CD.EXE sends the result of the request, which may contain large amounts of data, back to LOC-CD.SYS. At this point, LOC-CD.SYS returns the result to MSCDEX as if the data had come from a device dependant driver. In this way, the DOS in the local PC will accept LOC-CD.SYS as a valid device driver for an internal CD-ROM drive.

Chapter 4

Design of Net-CD Software

4.1 Operation of MSCDEX

Microsoft's CD Extensions (MSCDEX) require the presence of a CD-ROM device driver. This driver must be able to properly handle CD-ROM requests, while still meeting the requirements of a DOS device driver. Except for the initialization stage, all requests of the device driver are made through MSCDEX. These requests are in the DOS standard form of the *request header.* This request header varies in length dependant on the type of request made. The type of request is specified in the *command* field of the request header. The commands that are supported by LOC-CD.SYS are listed in Table 4.1. All commands with an index of 128 or greater are CD-ROM device driver specific commands.

Table 4.1: Driver Commands Supported by Net-CD

Index	Command
0	Init
3	IO Control Input
7	IO Control Flush
12	IO Control Output
13	Device Open
14	Device Close
128	Read Long
130	Read Long Prefetch
131	Seek

Normal operation of a CD-ROM drive consists of DOS making calls to MSCDEX which, in turn, makes requests of the CD-ROM device driver. However, DOS allows software to make direct requests of the device driver via its *multiplex interrupt* (INT 0x2F). These requests have a form and function identical to the request header. The multiplex interrupt will be shown to play a key role in the operation of REM-CD.EXE.

4.2 Design of LOC-CD.SYS

In order to implement LOC-CD.SYS, the source code for this driver must conform to DOS device driver requirements. Every DOS device driver must begin with the device header. This 13 byte header is used to inform DOS of the characteristics of the device, as well as pointers to two required procedures: *strategy* and *interrupt*. When DOS makes requests of the device, it first calls the strategy procedure, then follows with a call to the interrupt procedure.

Beyond the DOS device driver requirements, LOC-CD.SYS must be able to handle the extended CD-ROM device driver commands of MSCDEX. These commands, along with all supported regular commands except for initialization, must be sent to the remote CD-ROM site via the serial communications channel. This requires serial communication routines and a protocol for executing the command remotely. The baud rate for the serial line for this application is set to 115.2 kilobits per second, which is the maximum rate that DOS allows.

4.3 Communication Protocol

Table 4.2 depicts the protocol used for the Net-CD's data transmission. The local system begins the communication with a wakeup signal. After this signal is received and acknowledged by REM-CD.EXE, the system then begins a loop which begins when the local system's software receives a CD-ROM request from MSCDEX. The request is then sent to the remote system. The remote system then performs the MSCDEX request. The results of the request are then transmitted back to LOC-CD.SYS which, in turn, gives the results to MSCDEX. When an error occurs, the process detecting the error will transmit the error byte until the offending system returns the error acknowledge signal. At this point, the protocol returns to the original transmission of the request header length (denoted by a "*" in Table 4.2).

Table 4.2: Protocol for Net-CD Data Transmission

	Local System	Remote System
	Send Wakeup	
		Receive Wakeup
		Send Acknowledge(ACK)
	Receive ACK	
*	Get Request from MSCDEX	
**	Send Request Header (RH) Length	
		Receive RH Length
		Send ACK
	Receive ACK	
	Send RH Data Packet (RHP)	
		Receive RHP
		Send ACK
	Receive ACK	
		Perform Request
		Send RH Result Length
	Receive RH Result Length	
	Send ACK	
		Receive ACK
		Send RH Result Data Packet
	Receive RH Result Data Packet	
	Send ACK	
		Receive ACK
		Send CD Data Length
	Receive CD Data Length	
	Send ACK	
		Receive ACK
		Send Data Packet
	Receive Data Packet	
	Send ACK	
		Receive ACK
	Process Data	
	Return to *	Return to *

4.4 Design of REM-CD.EXE

REM-CD.EXE must be able to receive data via the serial line and execute commands sent from LOC-CD.SYS via the given protocol. Since the requests of MSCDEX to LOC-CD.SYS that are received by REM-CD.EXE are in a form standard to CD-ROM device driver requests, REM-CD.EXE can take advantage of the multiplex interrupt mentioned earlier to call the actual CD-ROM device driver, DRIVER.SYS, with ease.

Once DRIVER.SYS has processed the request, REM-CD.EXE can then send the results back to LOC-CD.SYS via the serial line. REM-CD.EXE is simply an interface between LOC-CD.SYS and the DRIVER.SYS. All data associated with each request returned according to the protocol, and is unchanged by REM-CD.EXE.

Chapter 5

Results

The implementation of Net-CD described in the previous chapters is completed and functional. The project was a success. Except for the slow CD-ROM data transfer delay, the CPE was indistinguishable from a PC containing a CD-ROM drive. Several CD-ROM applications were tested, both from DOS and from the Microsoft Windows environment. Files and directories were tested for integrity via a comparison between the information obtained by accessing the CD-ROM drive both locally and remotely.

The Net-CD data transfer rate on the serial line is 13.2 times slower than would be possible by using an ADSL line instead of a serial one. Assuming that a normal CD-ROM drive has a data transfer bandwidth of 300kbits/sec, the transfer of one data sector on a CD-ROM (2048 bytes) will take 54.6 milliseconds on a PC with a local CD-ROM drive. One sector of data will take 10.8 milliseconds to travel on an ADSL line, yielding a 19.8% increase in total data access time for one sector on the Net-CD system. The system implemented in this paper, however, has a data sector transfer time of 142 ms -- a 260% increase in data access time.

The 19.8% increase in data access time is not insignificant. This can be reduced, however, by copying the information on a CD-ROM to a large high-speed magnetic disk which could actually make the Net-CD faster to use than a local CD-ROM drive, and would allow multiple users to simultaneously access a single copy of the application.

Chapter 6

Conclusion

The networking of PC CD-ROM drives is one of many possible types of multimedia systems. The success of the Net-CD system described in this paper shows that PC CD-ROM networking is feasible. The implementation of a Net-CD, perhaps over a VDT type system, will give multimedia users access to a large library of multimedia applications and will help open the market for Video Dial Tone and other interactive multimedia systems

Appendix A

LOC-CD.ASM

```
.286

COM1 = 03F8H
COM2 = 02F8H
COM3 = 03E8H
COM4 = 02E8H

COM = COM1

; RETURNS ZERO FLAG TRUE (JZ WILL BE TAKEN) IF OK TO XMIT
X_OK MACRO
    MOV DX,COM+5
    IN AL,DX
    AND AL,020H
    CMP AL,020H
ENDM


; RETURNS ZERO CONDITION TRUE (JZ BRANCHES) IF OK TO READ REC.
    REGISTER
R_OK MACRO
    MOV DX,COM+5
    IN AL,DX
    AND AL,01H
    CMP AL,01H
    ENDM

X_RDY MACRO
; MOV AL,'*'
; CALL P_AL
 MOV DX,COM
 MOV AL,RCV_READY
 OUT DX,AL
 ENDM
```

```
R_RDY MACRO
 MOV DX,COM
 IN AL,DX
 CALL P_BYTE
 CMP AL,RCV_READY
 ENDM

RECV_ALM MACRO
 MOV DX,COM
 IN AL,DX
 ENDM

R_WAITM MACRO
@@: R_OK
 JNE @B
 ENDM



_TEXT SEGMENT BYTE PUBLIC 'CODE'
    ASSUME CS:_TEXT

;
; DEVICE DRIVERS ORIGINATE AT 0 (NOT 100H)
;
    ORG 0

VERY_TOP LABEL BYTE
;
; FIRST CHARACTER DEVICE HEADER (MUST BE AT OFFSET 0)

;
DEVICEHEADER:
    DW -1, -1 ;POINTER TO NEXT DEVICE
    DW 0C800H ;(CHARACTER, IOCTL, OPEN/CLOSE/RM)
    DW STRATEGY ;STRATEGY ROUTINE OFFSET
    DW INTERRUPT ;INTERRUPT ROUTINE OFFSET
    DB 'MSCD000 ' ;DEVICE NAME (MUST BE 8 VALID CHARS)
    DW 0 ;RESERVED
```

```
    DB 0 ;DRIVE LETTER
UNITS DB 1 ;NUMBER OF UNITS

REQUESTHEADER:
    DD 0 ;STRATEGY KEEPS REQUEST HEADER PTR HERE

; REQUEST HEADER LENGTH TABLE
RHLT:
    DB 23 ; ; 0-INIT
    DB 13 ; ; 1-ERROR
    DB 13 ; ; 2-ERROR
    DB 26 ; ; 3-IOCTL INPUT
    DB 13 ; ; 4-ERROR
    DB 13 ; ; 5-ERROR
    DB 13 ; ; 6-ERROR
    DB 13 ; ; 7-IOCTL FLUSH
    DB 13 ; ; 8-ERROR
    DB 13 ; ; 9-ERROR
    DB 13 ; ;10-ERROR
    DB 13 ; ;11-ERROR
    DB 26 ; ;12-IOCTL OUTPUT
    DB 13 ; ;13-DEVICE OPEN
    DB 13 ; ;14-DEVICE CLOSE

    DB 27 ; ;128-READ LONG
    DB 13 ; ;129-ERROR
    DB 27 ; ;130-READ LONG PREFETCH
    DB 24 ; ;131 SEEK
    DB 22 ; ;132 PLAY AUDIO
    DB 13 ; ;133-STOP AUDIO
    DB 13 ; ;134-ERROR
    DB 13 ; ;135-ERROR
    DB 13 ; ;136-RESUME AUDIO

PROCESSING_MSG DB 'PROCESSING MSCDEX REQUEST.',0DH,0AH,'$'
SENDING_MSG DB 'SENDING PACKET...','$'
SENDING_MSG2 DB 'SENDING RCV_READY','$'
SENT_MSG DB 'PACKET SENT',0DH,0AH,'$'
RECEIVED_MSG DB 'RCVED.',0DH,0AH,'$'
RCVING_MSG DB 'RCVING.','$'
```

```
SENT_BYTE DB 'BYTE SENT...','$'
RECEIVED_BYTE DB 'BYTE RECEIVED...','$'
OUT_DATA_MSG DB '(OUTDATA)','$'
DONE_MSG DB 'DONE WITH REQUEST',0DH,0AH,'$'
IOCTL_MSG DB 'IOCTL INPUT REQUEST #','$'
GETHDR DB 'GET HEADER ADDRESS',0DH,0AH,'$'
OTHERMSG DB 'OTHER OPERATION',0DH,0AH,'$'
EXITMSG DB 'EXITING OPERATION',0DH,0AH,'$'
HEXES DB '0123456789ABCDEF'
INTMSG DB 'INTERRUPT',0DH,0AH,'$'
STRMSG DB 'STRATEGY',0DH,0AH,'$'
DBG_MSG DB 'ERROR! OFFSET!=0','$'

OK_MSG DB "LOC-CD READY.",0AH,0DH,'$'
COMMAND DB 8
DATA_LEN DW ?
DTA_OFS DW ?
DTA_SEG DW ?
DOS_DTA_OFS DW ?
DOS_DTA_SEG DW ?
TMP_COUNT DW 3


;============================================================
    ===========
;
; STRATEGY
;

; DEVICE STRATEGY ROUTINE. SAVES THE POINTER TO THE CALLER'S
    REQUEST
; HEADER STRUCTURE FOR THE SUBSEQUENT CALL INTO INTERRUPT
    BELOW.
;
; ENTRY:
; ES:BX - FAR PTR TO THE CALLER'S REQUEST HEADER.
;
; EXIT:
; REQUESTHEADER - FAR PTR TO THE CALLER'S REQUEST HEADER.
;
;============================================================
```

```
    ===========

PR_RH PROC NEAR


 PUSH AX
 PUSH BX
PRH_LOOP:
 MOV AX,BX
 CALL P_BYTE
 MOV AL,':'
 CALL PRNT_AL
 MOV AL,BYTE PTR DS:[BX]
 CALL P_BYTE
 INC BX
 MOV AL,' '
 CALL PRNT_AL
 CALL PRNT_AL
 LOOP PRH_LOOP
 POP BX
 POP AX
 RET
 PR_RH ENDP



SAVE_XFER PROC NEAR
 PUSH AX
 MOV AX,WORD PTR DS:[BX+14]
 MOV CS:DOS_DTA_OFS,AX
 CMP AX,0
 JE @F
 MOV DI,OFFSET CS:DBG_MSG
 CALL PR_MSG
@@:
 MOV AX,WORD PTR DS:[BX+16]
 MOV CS:DOS_DTA_SEG,AX
 POP AX
 RET
SAVE_XFER ENDP
```

```
RESTORE_XFER PROC NEAR
 PUSH AX
 MOV AX,WORD PTR CS:DOS_DTA_OFS
 MOV CS:DTA_OFS,AX
 MOV WORD PTR DS:[BX+14],AX
 MOV AX,CS:DOS_DTA_SEG
 MOV CS:DTA_SEG,AX
 MOV WORD PTR DS:[BX+16],AX
 POP AX
 RET
RESTORE_XFER ENDP

PRINT_DATA PROC NEAR
 PUSH CX
 PUSH BX
 PUSH AX
 PUSH ES
 MOV AX,CS:DOS_DTA_SEG
 MOV ES,AX
 MOV BX,CS:DOS_DTA_OFS
 MOV CX,256
PDATA_LOOP:
 MOV AL,BYTE PTR ES:[BX]
 CALL P_BYTE
 MOV AL,' '
 CALL PRNT_AL
 CALL PRNT_AL
 INC BX
 LOOP PDATA_LOOP
 POP ES
 POP AX
 POP BX
 POP CX
 RET
PRINT_DATA ENDP

OUT_DATA PROC NEAR
 PUSH BX
 PUSH ES
```

```
 PUSH CX
 PUSH AX
; PUSH DI
; MOV DI,OFFSET CS:OUT_DATA_MSG
; CALL PR_MSG
 MOV CX,128
 MOV ES,CS:DOS_DTA_SEG
 MOV BX,CS:DOS_DTA_OFS
 MOV AH,0
@@:
 MOV AL,BYTE PTR ES:[BX]
 CALL P_BYTE
 INC BX
 LOOP @B
 POP DI
 POP AX
 POP CX
 POP ES
 POP BX
 RET
OUT_DATA ENDP




;
; ON ENTRY:
; DS:BX POINTS TO REQUEST HEADER

SEND_REQUEST_HEADER PROC NEAR
 MOV CS:COMMAND,RHP
 PUSH DI
 PUSH AX
 MOV AL,DS:RQH_CMD[BX]
 CMP AL,14
 JBE @F
 SUB AL,113
@@:
 MOV AH,0
```

```
 MOV DI,AX
 MOV AL,BYTE PTR CS:RHLT[DI]
 MOV AL,BYTE PTR DS:[BX]
 MOV CS:DATA_LEN,AX
 MOV CS:DTA_OFS,BX
 MOV CS:DTA_SEG,DS
 CALL SND_PKT
 CALL SAVE_XFER
 MOV AL,DS:[BX].RQH_CMD

; MOV CX,CS:DATA_LEN
; CALL PR_RH

 CMP AL,3
 JNE @F
 ; IOCTL_I
    LES DI,IOCTL_XFER[BX] ;APPLICATION'S TRANSFER ADDRESS
 MOV CS:DTA_OFS,DI
 MOV CS:DTA_SEG,ES
 MOV CS:DATA_LEN,1
 CALL SND_PKT
@@: CMP AL,12
 JNE @F
    LES DI,IOCTL_XFER[BX] ;APPLICATION'S TRANSFER ADDRESS
 MOV CS:DTA_OFS,DI ; IOCTL_O
 MOV CS:DTA_SEG,ES
 MOV CS:DATA_LEN,1
 CALL SND_PKT
@@:
 POP AX
 POP DI
    RET

SEND_REQUEST_HEADER ENDP

RECEIVE_REQUEST_HEADER PROC NEAR
 PUSH AX
 PUSH DI
 MOV CS:DTA_OFS,BX
 MOV CS:DTA_SEG,DS
```

```
 CALL RCV_PKT
 CALL RESTORE_XFER
 CALL RCV_PKT
 MOV AX,CS:DATA_LEN
 CMP AX,256
 JB @F
 CALL PRINT_DATA
@@:
 POP DI
 POP AX
 RET
RECEIVE_REQUEST_HEADER ENDP


STRATEGY PROC FAR

    MOV WORD PTR CS:REQUESTHEADER.LO,BX
    MOV WORD PTR CS:REQUESTHEADER.HI,ES
 MOV DI,OFFSET CS:STRMSG
 CALL PR_MSG
    RET

STRATEGY ENDP




;============================================================
    ===========
;
; INTERRUPT
;
; MAIN ENTRY POINT TO THE DEVICE INTERRUPT HANDLER.
;
; ENTRY:
; REQUESTHEADER - FAR PTR TO THE CALLER'S REQUEST HEADER.
; (SEE MSCDEX.INC) FOR FORMAT OF THE REQUEST HEADER)
;
;============================================================
    ===========
INTERRUPT PROC FAR
```

```
    PUSH AX
    PUSH BX
    PUSH CX
    PUSH DX
    PUSH SI
    PUSH DI
    PUSH BP
    PUSH DS
    PUSH ES
 MOV DI,OFFSET CS:INTMSG
 CALL PR_MSG
 CALL SET_BD
;
; POINT DS:BX AT THE REQUEST HEADER
;
    LDS BX,DWORD PTR CS:REQUESTHEADER

    MOV AL,DS:[BX].RQH_CMD ;GET COMMAND
    CMP AL,DVRQ_NCMD_MAX ;EXTENDED COMMAND?
    JBE OK_CMD ; NOPE
;
; MUST BE EXTENDED CDROM DEVICE DRIVER COMMAND, CONVERT INTO
    INDEX AT
; END THE NORMAL CDROM DEVICE DRIVER FUNCTION TABLE
;
    CMP AL,DVRQ_ECMD_MIN
    JB @F
    SUB AL,(DVRQ_ECMD_MIN - DVRQ_NCMD_MAX - 1)
    CMP AL,(DVRQ_NCMD_MAX + DVRQ_ECMD_MAX - DVRQ_ECMD_MIN +
    1)
    JBE OK_CMD ; YES
@@:
    JMP ERROR ; NOPE, COMMAND OUT OF RANGE

OK_CMD:
    CBW
    OR AX,AX ;INIT TIME?
    JZ INIT ; YES
    ; NO, DISPATCH TO COMMAND
    CMP AX,3
```

```
    JE IOCTL_I

SEND:
 MOV DI,OFFSET CS:PROCESSING_MSG
 CALL PR_MSG
    CALL SEND_REQUEST_HEADER
 CALL RECEIVE_REQUEST_HEADER
    JMP EXIT_INIT


;============================================================
    ===========
;
; INIT
;
; INIT, CD-ROM DEVICE DRIVER ROUTINE TO INITIALIZE THE DRIVER.
;
; THIS IS THE ONLY DEVICE DRIVER CALL COMING DIRECTLY FROM
    DOS, AND IS
; ONLY MADE ONCE. INIT SHOULD INITIALIZE CERTAIN FIELDS IN THE
    REQUEST
; HEADER:
;
; - SET INIT_UNITS AND INIT_DEVNO TO 0 SINCE DOS VIEWS THIS
    AS A CHARACTER
; DEVICE (MSCDEX MAKES ITS OWN DETERMINATION OF THE NUMBER OF
    UNITS THROUGH
; THE DEVICE HEADER).
;
; - RETURN THE ADDRESS OF THE END OF THE RESIDENT CODE/DATA
    SECTION IN
; INIT_ENDADDR. CODE/DATA AFTER THIS POINTER IS DISCARDED.
;
; - PARSE THE CONFIG.SYS LINE AFTER THE '=' CHARACTER (POINTED
    TO BY
; INIT_BPBARR) FOR THE DEVICE NAME AND FILL IN THE DEV_NAME
    FIELD IN
; THE DEVICE HEADER MAKING SURE IT IS A LEGAL 8-CHARACTER
    FILENAME
; (PADDED OUT TO 8 CHARACTERS WITH SPACES IF NECESSARY).
;
```

```
; ENTRY:
; DS:BX - FAR PTR TO THE REQUEST HEADER
; INITHEADER STRUC
;
; INIT_RQH DB SIZE REQUEST_HDR DUP (?)
; INIT_UNITS DB ? -> 0
; INIT_ENDADDR DD ? -> VERY_END
; INIT_BPBARR DD ?
; INIT_DEVNO DB ? -> 0
;
; INITHEADER ENDS
;
;===========================================================
    ===========


INIT:
    MOV DS:[BX].INIT_UNITS,0
    MOV DS:[BX].INIT_DEVNO,0
    MOV WORD PTR DS:[BX+2].INIT_ENDADDR,CS
    MOV WORD PTR DS:[BX].INIT_ENDADDR,OFFSET VERY_END
    JMP EXIT_INIT


;_IOCTLHEADER STRUC
;IOCTL_RQH DB SIZE _REQUESTHEADER DUP (?)
;IOCTL_MEDIA DB ?
;IOCTL_XFER DD ?
;IOCTL_NBYTES DW ?
;IOCTL_SECTOR DW ?
;IOCTL_VALID DD ?
;_IOCTLHEADER ENDS

IOCTL_I:
    LES DI,IOCTL_XFER[BX] ;APPLICATION'S TRANSFER ADDRESS
    MOV AL,BYTE PTR ES:[DI]
 MOV AX,0
    CMP AX,0
    JNE SEND
    INC DI
```

```
    MOV WORD PTR ES:[DI],OFFSET CS:DEVICEHEADER
    MOV WORD PTR ES:[DI+2],CS
    JMP EXIT_NOT_BUSY



;============================================================
    =
;
; IOCTL INPUT COMMANDS...
;
;============================================================
    =

;------------------------------------------------------------
    -----------
;
; RETURN_ADDRESS
;
; IOCTL INPUT SUB-FUNCTION #0.
;
; RETURN THE ADDRESS OF THE DEVICE HEADER.
;
;------------------------------------------------------------
    -----------
RETURN_ADDRESS:


    MOV WORD PTR ES:[DI].IO_DEVADDR,OFFSET DEVICEHEADER
    MOV WORD PTR ES:[DI+2].IO_DEVADDR,CS
; JMP SHORT IOCTL_IO_EXIT


;============================================================
    ===========
;
; DEVICE_OPEN
;
; DEVICE OPEN, CD-ROM DEVICE DRIVER ROUTINE INDICATING TO THE
    DEVICE
```

```
; DRIVER THAT AN APPLICATION IS BEGINNING TO USE IT.
;
;============================================================
    ===========

;------------------------------------------------------------
    -----------
;
; EXIT POINTS.
;
;------------------------------------------------------------
    -----------

ERROR:
;
; EXIT INDICATING UNKNOWN COMMAND ERROR
;
    MOV AX,(ERRBIT + DONEBIT + DRVERR_UNKNOWN_COMMAND)
    JMP SHORT EXIT


EXIT_BUSY:
;
; EXIT INDICATING NO ERRORS AND AUDIO IS PLAYING
;
    MOV AX,(DONEBIT + BUSYBIT)
    JMP SHORT EXIT



EXIT_NOT_BUSY:
;
; EXIT INDICATING NO ERRORS AND AUDIO IS NOT PLAYING
;
    MOV AX,DONEBIT
    LDS BX,DWORD PTR CS:REQUESTHEADER
    MOV [BX].RQH_STATUS,AX


EXIT:
;
```

```
; EXIT AND POKE THE STATUS WORD INTO THE REQUEST HEADER.
; THIS IS ALWAYS THE FINAL EXIT POINT.
;
;   LDS BX,DWORD PTR CS:REQUESTHEADER
;   MOV [BX].RQH_STATUS,AX


EXIT_INIT:
;
; EXIT POINT FROM INIT
;
 MOV DI,OFFSET CS:DONE_MSG
 CALL PR_MSG
    POP ES
    POP DS
    POP BP
    POP DI
    POP SI
    POP DX
    POP CX
    POP BX
    POP AX
    RET

INTERRUPT ENDP

DELAY PROC NEAR
 PUSH CX
 MOV CX,0FFFFH
@@:
 JMP SHORT $+2
 LOOP @B
 POP CX
 RET

DELAY ENDP
```

```
; FUNCTIONS IN SERIAL.ASM
; MACRO X_OK,R_OK => XMIT OR RCV OK TEST. JZ WILL BRANCH IF OK
;
; PROC SET_BD => SET BAUD RATE TO 1200 BAUD, 8N1
; PROC XMIT_AL => PLACE [AL] IN XMIT REGISTER
; PROC X_WAIT => LOOP UNTIL OK TO XMIT
; PROC RECV_AL => READ RCV REGISTER TO AL
; PROC R_WAIT => LOOP UNTIL OK TO READ RCV REGISTER
; PROC PRNT_AL => PRINTS THE ASCII REPRESENTATION OF AL TO
   CONSOLE
; PROC R_FLUSH => FLUSHES RCV BUFFERS



SET_BD PROC NEAR

    MOV AL,080H ; SET DLAB IN REGISTER 3 OF COM1
    MOV DX,COM+3
    OUT DX,AL
    JMP SHORT $+2

    MOV AL,01H ; SET BAUD LSB TO 1 (REG. 0)
    MOV DX,COM
    OUT DX,AL
    JMP SHORT $+2

    MOV AL,0 ; SET BAUD MSB TO 0 (REG. 1)
    MOV DX,COM+1
    OUT DX,AL
    JMP SHORT $+2

    MOV AL,03H ; SET DLAB=0, REG. 3 = 8-N-1
    MOV DX,COM+3
    OUT DX,AL
    JMP SHORT $+2
    RET
```

```
SET_BD ENDP


XMIT_AL PROC NEAR

    MOV DX,COM
    OUT DX,AL
; CALL P_BYTE
    RET

XMIT_AL ENDP

X_WAIT PROC NEAR

    PUSH AX
TBE:
    X_OK
    JNE TBE
    POP AX
    RET
X_WAIT ENDP




RECV_AL PROC NEAR

    MOV DX,COM
    IN AL,DX
    RET



RECV_AL ENDP
P_AL PROC NEAR
 PUSH AX
 PUSH BX
    MOV AH,0EH
```

```
    MOV BH,0
    ;INT 10H
 POP BX
 POP AX
    RET
P_AL ENDP

PRNT_AL PROC NEAR
; PUSH AX
; PUSH BX
;   MOV AH,0EH
;   MOV BH,0
;   INT 10H
; POP BX
; POP AX
    RET
PRNT_AL ENDP

; PRINT MESSAGE PROCEDURE
; ON ENTRY:
; DI POINTS TO MESSAGE IN THE DATA
; SEGMENT WHICH IS TERMINATED BY A $

PR_MSG PROC NEAR
    PUSH AX
    PUSH BX
MSGLOOP:
    MOV AL,CS:[DI]
    CMP AL,'$'
    JE MSGDONE
    PUSH DI
    CALL P_AL
    POP DI
    INC DI
    JMP MSGLOOP
MSGDONE:
    POP BX
    POP AX
    RET
PR_MSG ENDP
```

```
; P_WORD PRINTS THE VALUE IN AX AS A HEXADECIMAL
P_WORD PROC NEAR
 PUSH AX
 CALL P_BYTE
 SHR AX,8
 CALL P_BYTE
 POP AX
 RET
P_WORD ENDP


; P_BYTE PRINTS THE VALUE IN AL AS A HEXADECIMAL
P_BYTE PROC NEAR
 PUSH BX
 PUSH AX

 MOV BX,AX
 SHR BX,4
 AND BX,000FH
 MOV AL,CS:HEXES[BX]
 CALL P_AL
 POP BX
 PUSH BX
 AND BX,000FH
 MOV AL,CS:HEXES[BX]
 CALL P_AL
 POP AX
 POP BX
 RET
P_BYTE ENDP
```

```
R_WAIT PROC NEAR

; MOV AL,'+'
; CALL P_AL
RXRDY:
    R_OK
    JNE RXRDY
    RET

R_WAIT ENDP


R_FLUSH PROC NEAR

FLUSHER:
    R_OK
    JNE FL_DONE
    CALL RECV_AL
 JMP FLUSHER
FL_DONE:
    RET

R_FLUSH ENDP

ONE_FLUSH PROC NEAR
 PUSH CX
 MOV CX,10
@@:
 R_OK
 JNE @F
 CALL RECV_AL
 LOOP @B
@@:
 POP CX
    RET

ONE_FLUSH ENDP

RCV_PKT PROC NEAR
```

```
    PUSH ES
    PUSH DI
    PUSH AX
    PUSH BX
    PUSH CX
 CALL R_FLUSH
 MOV DI,OFFSET CS:SENDING_MSG2
 CALL PR_MSG

 MOV AX,CS:DTA_OFS
 SHR AX,4
 ADD AX,CS:DTA_SEG
 MOV CS:DTA_SEG,AX
 MOV AX,CS:DTA_OFS
 AND AX,000FH
 MOV CS:DTA_OFS,AX

    MOV AX,CS:DTA_SEG
 MOV ES,AX
 MOV AX,CS:DTA_OFS
    MOV DI,AX

 CALL R_FLUSH
RCV_GO:
 MOV AL,'-'
 CALL P_AL

SND_CHK:
 CALL X_WAIT
 X_RDY
 MOV CX,65535
@@:
 R_OK
 JE RCV
 LOOP @B
 R_OK
 JNE SND_CHK
RCV:

 RECV_ALM
```

```
 CMP AL,CS:COMMAND
 JNE RCV_GO
 CLI
    R_WAITM
    RECV_ALM
    MOV BYTE PTR CS:DATA_LEN[0],AL
    R_WAITM
    RECV_ALM
    MOV BYTE PTR CS:DATA_LEN[1],AL


    MOV CX,CS:DATA_LEN
    CMP CX,0
    JE RCV_DONE

RCV_LOOP:
    R_WAITM
    RECV_ALM
    MOV BYTE PTR ES:[DI],AL
    INC DI
    LOOP RCV_LOOP

RCV_DONE:
 STI
 MOV DI,OFFSET CS:RECEIVED_MSG
 CALL PR_MSG
    POP CX
    POP BX
    POP AX
    POP DI
    POP ES
    RET
RCV_PKT ENDP

SND_PKT PROC NEAR
    PUSH ES
    PUSH DI
    PUSH AX
    PUSH BX
    PUSH CX
```

```
 CALL DELAY
 MOV AX,CS:DTA_OFS
 SHR AX,4
 ADD AX,CS:DTA_SEG
 MOV CS:DTA_SEG,AX
 MOV AX,CS:DTA_OFS
 AND AX,000FH
 MOV CS:DTA_OFS,AX
 MOV DI,OFFSET CS:SENDING_MSG
 CALL PR_MSG
@@: CALL R_WAIT
 R_RDY
 JNE @B
 MOV DI,OFFSET CS:OK_MSG
 CALL PR_MSG
    MOV AL,CS:COMMAND
 CALL P_BYTE
 CALL DELAY
 CALL X_WAIT
 CALL XMIT_AL
    MOV AL,BYTE PTR CS:DATA_LEN[0]
 CALL DELAY
    CALL X_WAIT
    CALL XMIT_AL
    MOV AL,BYTE PTR CS:DATA_LEN[1]
 CALL DELAY
    CALL X_WAIT
    CALL XMIT_AL
 MOV AX,CS:DATA_LEN
 CALL P_WORD
 CALL ONE_FLUSH
    MOV CX,CS:DATA_LEN
    CMP CX,0
    JE SND_DONE
    MOV ES,CS:DTA_SEG
    MOV DI,CS:DTA_OFS
SND_LOOP:
 CALL DELAY
 CALL X_WAIT
    MOV AL,BYTE PTR ES:[DI]
```

```
    CALL XMIT_AL
 INC DI
    LOOP SND_LOOP

SND_DONE:
 MOV DI,OFFSET CS:SENT_MSG
 CALL PR_MSG
    POP CX
    POP BX
    POP AX
    POP DI
    POP ES
    RET
SND_PKT ENDP

VERY_END:
_TEXT ENDS
 END
```

Appendix B

REM-CD.ASM

```
; Rem-CD.asm
   ;
   ; Remote executable to send data
   ; and interpret requests for the
   ; Net-CD system.
   ;
   ; Steve Levis
   ; 12/19/94

   .286
   .MODEL LARGE



   COM1=03F8h
   COM2 = 02F8h
   COM3 = 03E8h
   COM4=02E8h

   COM=COM4

   INCLUDE CMDS.ASM

   ; Returns Zero Flag True (JZ will be taken) if OK to xmit
   X_OKMACRO
      MOVDX,COM+5
      INAL,DX
      ANDAL,020h
      CMPAL,020h
   ENDM


   ; Returns Zero Condition True (JZ Branches) if OK to read
rec. register
   R_OKMACRO
      MOVDX,COM+5
```

```
INAL,DX
ANDAL,01h
CMPAL,01h
ENDM

X_RDY MACRO
; MOV AL,'*'
; CALL P_AL
MOV DX,COM
MOV AL,Rcv_Ready
OUT DX,AL
; CALL P_BYTE
ENDM

R_RDY MACRO
MOV DX,COM
IN AL,DX
CMP AL,Rcv_Ready
ENDM

RECV_ALM MACRO
MOV DX,COM
IN AL,DX
ENDM

R_WAITM MACRO
; PUSH BX
; MOV CX,0FFFFh
; MOV BX,0
@@: R_OK
JNE @B
; JE @F
; LOOP @B
; MOV BX,1
;@@:
;
; CMP BX,0
; POP BX
ENDM
```

```
; Functions in SERIAL.ASM
; MACRO X_OK,R_OK => Xmit or Rcv OK test.JZ  will  branch
if ok
;
; PROCSET_BD => Set Baud Rate to 1200 Baud, 8N1
; PROCXMIT_AL => Place [AL] in Xmit Register
; PROCX_WAIT => Loop until OK to Xmit
; PROCRECV_AL => Read Rcv Register to AL
; PROCR_WAIT => Loop until OK to read Rcv Register
; PROCPRNT_AL => Prints the ASCII representation of AL to
console
; PROCR_FLUSH => Flushes Rcv Buffers

SSEGSEGMENT STACK
  DB32 DUP("STACK---")
SSEGENDS

DSEGSEGMENT
OK_MSGDB"Ok.",'$'
command db 8
data_len dw 31
DTA_ofs dw ?
DTA_seg dw ?
dbg_msg db "Non-Cooked sector requested!",'$'
RH_MSG db "Request Header Command:",'$'
Special_MSG db "No special packet required",0ah,0dh,'$'
sent_msg db "Packet sent.",0ah,0dh,'$'
sending_msg db "Sending packet...",'$'
received_msg db "Packet received.",0ah,0dh,'$'
rcving_msg db "Receiving packet...",'$'
waiting_msg db "Waiting for packet...",'$'
received_byte db "Byte received...",'$'
sent_byte db "Byte sent...",'$'
cmd_msg db "Cmd Rec.",'$'
hexes db '0123456789ABCDEF'
unx_msg db 'Unexpected command:','$'
s_req_msg db 'Sending Request',0ah,0dh,'$'
s_rh_msg db 'Sending Request Header',0ah,0dh,'$'
delay_msg db' $'
```

```
s_data_msg db 'Sending Data',0ah,0dh,'$'
inting_msg db 'Calling Interrupt 2F','$'
newline db 0ah,0dh,'$'
ReqHdr db 2 Dup("Request Header__")
RH_Len dw ?
DT_Len dw ?
DSEGENDS


BSEG1 SEGMENT
ORG 0
Buffer1 db 8192 DUP("BUFFER__")
BSEG1 ENDS

CSEGSEGMENT 'CODE'
ASSUMECS:CSEG,DS:DSEG,SS:SSEG

MAINPROCFAR


   PUSHDS
   PUSH0
   MOVAX,DSEG
   MOVDS,AX


   CALLSET_BD
   CALLR_FLUSH
 MOV DI,OFFSET OK_MSG
 CALL PR_MSG

PROC_CMD:
 MOV COMMAND,RHP
 CALL RCV_RH
 CMP DT_LEN,63488
 JA ENDMAIN
 MOV DI,OFFSET DS:S_REQ_MSG
 CALL PR_MSG
 CALL SND_REQ
 MOV DI,OFFSET DS:S_RH_MSG
```

```
 CALL PR_MSG
 CALL SND_RH
 MOV DI,OFFSET DS:S_DATA_MSG
 CALL PR_MSG
 CALL SND_DATA
 JMP PROC_CMD

ENDMAIN:
 MOV AX,DT_LEN
 CALL P_WORD
 RET

MAINENDP

delay proc near
 push cx
 push di
 mov di,offset ds:delay_msg
 call pr_msg
 mov cx,0FFFFh
@@:
 jmp short $+2
 loop @b
 pop di
 pop cx
 ret
delay endp


RCV_RH PROC NEAR
 MOV DTA_ofs,OFFSET ReqHdr
 MOV DTA_seg,DS
 CALL RCV_PKT
 MOV AX,DS:Data_Len
 MOV DS:RH_Len,AX

 MOV AL,DS:ReqHdr[2] ; Command Code Field
 CMP AL,3
 JNE @f
 ; IOCTL_I
```

```
 MOV WORD PTR DS:ReqHdr[14],OFFSET BSEG1:Buffer1
 MOV WORD PTR DS:ReqHdr[16],BSEG1
 MOV AX,WORD PTR DS:ReqHdr[18]
 MOV DT_LEN,AX
 MOV DTA_ofs,OFFSET BSEG1:Buffer1
 MOV DTA_seg,BSEG1
 CALL RCV_PKT
 JMP RH_DONE
@@:
 CMP AL,12
 JNE @f
 ; IOCTL_O
 MOV WORD PTR DS:ReqHdr[14],OFFSET BSEG1:Buffer1
 MOV WORD PTR DS:ReqHdr[16],BSEG1
 MOV AX,WORD PTR DS:ReqHdr[18]
 MOV DT_LEN,AX
 MOV DTA_ofs,OFFSET BSEG1:Buffer1
 MOV DTA_seg,BSEG1
 CALL RCV_PKT
 JMP RH_DONE

@@:
 CMP AL,128
 JNE @f


READ_LONG: ; READ LONG
 MOV WORD PTR DS:ReqHdr[14],OFFSET BSEG1:Buffer1
 MOV WORD PTR DS:ReqHdr[16],BSEG1
 MOV AX,WORD PTR DS:ReqHdr[18]
 SHL AX,11 ; AX = AX * 2048
 MOV DT_LEN,AX
 MOV AL,BYTE PTR DS:ReqHdr[24]
 CMP AL,0
 JE RLDONE
 MOV DI,OFFSET DS:DBG_MSG
 CALL PR_MSG
STAY: JMP STAY

RLDONE: MOV DTA_ofs,OFFSET BSEG1:Buffer1
```

```
 MOV DTA_seg,BSEG1

 JMP RH_DONE

@@:
 MOV DS:DT_LEN,0

RH_DONE:
 MOV AL,DS:ReqHdr[2]
 CALL P_BYTE
 MOV AL,' '
 CALL P_AL
 RET
RCV_RH ENDP

SND_REQ PROC NEAR
 MOV CX,50
@@: PUSHCX
 MOV AL,'#'
 CALL P_AL
 MOV AX,1510h
 MOV CX,4 ; Drive E:
 MOV BX,DS
 MOV ES,BX
 MOV bx,OFFSET BSEG1:Buffer1
 MOV word ptr DS:ReqHdr[14],bx
 MOV bx,BSEG1
 MOV word ptr DS:ReqHdr[16],Bx
 MOV BX,OFFSET DS:ReqHdr
 INT 02Fh
 MOV CX,word ptr DS:ReqHdr[3]
 AND CX,8300h
 CMP CX,0100h
 JE @F
 POP CX
 LOOP @B
 RET
@@:
 POP CX
 RET
```

```
SND_REQ ENDP

SND_RH PROC NEAR
 MOV DTA_ofs,OFFSET DS:ReqHdr
 MOV DTA_seg,DS
 MOV AX,DS:RH_Len
 MOV Data_Len,AX
 CALL SND_PKT
 RET
SND_RH ENDP

SND_DATA PROC NEAR
 MOV AX,DT_Len
 CALL P_WORD
 MOV Data_Len,AX
 MOV DTA_seg,BSEG1
 MOV DTA_ofs,OFFSET BSEG1:Buffer1
 CALL SND_PKT
 RET
SND_DATA ENDP


SET_BDPROCNEAR

   MOV AL,080h ; Set DLAB in Register 3 of COM1
   MOVDX,COM+3
   OUT DX,AL
   JMP SHORT $+2

   MOV AL,01h ; Set baud LSB to 1 (Reg. 0)
   MOVDX,COM
   OUT DX,AL
   JMP SHORT $+2

   MOV AL,0 ; Set baud MSB to 0 (Reg. 1)
   MOVDX,COM+1
   OUT DX,AL
   JMP SHORT $+2

   MOV AL,03h ; Set DLAB=0, Reg. 3 = 8-N-1
```

```
   MOVDX,COM+3
   OUT DX,AL
   JMP SHORT $+2
   RET

SET_BD ENDP


XMIT_AL PROC NEAR

; PUSH DI
; MOV DI,offset sent_byte
; CALL PR_MSG
; POP DI
   MOVDX,COM
   OUT DX,AL
   RET

XMIT_AL ENDP

X_WAIT PROC NEAR

 PUSH AX
TBE:
; MOV AL,'+'
; CALL P_AL
   X_OK
   JNETBE

 POP AX
   RET
X_WAITENDP




RECV_AL PROC NEAR

   MOVDX,COM
```

```
   IN AL,DX
   RET

RECV_AL ENDP

P_AL PROC NEAR
 PUSH AX
 PUSH BX
   MOV AH,0EH
   MOV BH,0
   INT 10h
 POP BX
 POP AX
   RET
P_AL ENDP

PRNT_AL PROC NEAR
; PUSH AX
; PUSH BX
;  MOV AH,0EH
;  MOV BH,0
;  INT 10h
; POP BX
; POP AX
   RET
PRNT_AL ENDP

; Print Message Procedure
; On Entry:
; DI Points to Message in the data
; segment which is terminated by a $

PR_MSG PROC NEAR
 PUSH AX
 PUSH BX
MSGLOOP:
 MOV AL,DS:[DI]
 CMP AL,'$'
 JE MSGDONE
 CALL P_AL
```

```
 INC DI
 JMP MSGLOOP
MSGDONE:
 POP BX
 POP AX
 RET
PR_MSG ENDP


; p_word prints the value in ax as a hexadecimal
p_word proc near
 push ax
 call p_byte
 shr ax,8
 call p_byte
 pop ax
 ret
p_word endp


; p_byte prints the value in al as a hexadecimal
p_byte proc near
 push bx
 push ax
 mov bx,ax
 shr bx,4
 and bx,000Fh
 mov al,ds:hexes[bx]
 call p_al
 pop bx
 push bx
 and bx,000Fh
 mov al,ds:hexes[bx]
 call p_al
 pop ax
 pop bx
 ret
p_byte endp
```

```
R_WAIT PROC NEAR

 PUSH AX
RxRDY:
   R_OK
   JNERxRDY
 POP AX
   RET

R_WAIT ENDP


R_FLUSH PROC NEAR

FLUSHER:
   R_OK
   JNEFL_DONE
   CALL RECV_AL
 JMP FLUSHER
FL_DONE:
   RET

R_FLUSH ENDP

ONE_FLUSH PROC NEAR

 R_OK
 JNE @F
 CALL RECV_AL
@@: RET

ONE_FLUSH ENDP

rcv_pkt proc near
 push es
 push di
 push ax
 push bx
 push cx
 MOV DI,OFFSET DS:waiting_msg
```

```
 CALL PR_MSG

 mov ax,CS:DTA_ofs
 shr ax,4
 add ax,CS:DTA_seg
 mov CS:DTA_seg,ax
 mov ax,CS:DTA_ofs
 and ax,000Fh
 mov CS:DTA_ofs,ax

 mov es,ds:DTA_seg
 mov di,ds:DTA_ofs
 call r_flush
 mov al,'-'
 call p_al
snd_chk: ;(or money order)
 call x_wait
 x_rdy
 mov cx,65535
@@:
 r_ok
 je rcv
 loop @b
 r_ok
 jne snd_chk

rcv:
 recv_alm
 cmp al,ds:command
 jne snd_chk
xfer_com:
 cli
 r_waitm
 recv_alm
 mov byte ptr ds:data_len[0],al
 r_waitm
 recv_alm
 mov byte ptr ds:data_len[1],al
 mov cx,ds:data_len
 cmp cx,0
```

```
 je rcv_done

rcv_loop:
 sti
 r_waitm
 cli
 recv_alm
 mov byte ptr es:[di],al
 inc di
 loop rcv_loop

rcv_done:
 sti
 mov di,offset received_msg
 call pr_msg
 pop cx
 pop bx
 pop ax
 pop di
 pop es
 ret
rcv_pkt endp

snd_pkt proc near
 push es
 push di
 push ax
 push bx
 push cx
 mov di,offset waiting_msg
 call pr_msg

 mov ax,CS:DTA_ofs
 shr ax,4
 add ax,CS:DTA_seg
 mov CS:DTA_seg,ax
 mov ax,CS:DTA_ofs
 and ax,000Fh
 mov CS:DTA_ofs,ax
```

```
@@: call r_wait
 r_rdy
 jne @B
 call delay
 mov di,offset ok_msg
 call pr_msg
 mov al,ds:command
 call delay
 call x_wait
 call xmit_al
 mov al,byte ptr ds:data_len[0]
 call delay
 call x_wait
 call xmit_al
 mov al,byte ptr ds:data_len[1]
 call delay
 call x_wait
 call xmit_al
 call one_flush
 mov cx,ds:data_len
 cmp cx,0
 je snd_done
 mov es,ds:DTA_seg
 mov di,ds:DTA_ofs
snd_loop:
 call x_wait
 mov al,byte ptr es:[di]
 call xmit_al
 inc di
 loop snd_loop

snd_done:
 mov di,offset sent_msg
 call pr_msg
 pop cx
 pop bx
 pop ax
 pop di
 pop es
 ret
```

```
snd_pkt endp

CSEGENDS
   ENDMAIN
```