A Knowledge-Based, Event-Driven, Real-Time Operating System for an ISDN Personal Workstation

Zenon Slodki

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Electrical Engineering at
Concordia University
Montréal, Québec, Canada

June 1990

ISBN 0-315-59162-5

ABSTRACT

A Knowledge-Based, Event-Driven, Real-Time Operating System for an ISDN Personal Workstation

Zenon Slodki

The concept of a personal computer is evolving as communications and computers merge. A growing demand in telecommunications is to provide access to external databases. This is being addressed with the development of the Integrated Services Digital Network (ISDN). To support such a service, improved processing is required from personal computers. A key type of processing necessary to manipulate copious amounts of information is the symbolic proc ssing that is part of a knowledge-based system.

Another application of computer communications is autonomous control of the home environment. By communicating with the electronic equipment within the home, control of these devices is possible. This capability requires well developed interfacing support from the computer.

In this thesis, we propose an operating system for an ISDN Personal Workstation that will meet the above mentioned demands. The operating system is described in detail and a version of it is implemented for our Experimental ISDN Personal Workstation. The operating system architecture is event-driven in order to support autonomous control. The architecture also incorporates a knowledge-based system and provides real-time support.

Acknowledgements

I would like to thank my thesis supervisor, Dr.Tho Le-Ngoc for his guidance throughout this research and for his advice and constructive criticism during the preparation of this thesis.

I would also like to thank Mr.Robert Rourke for his assistance, support and friendship throughout my graduate studies. The countless discussions over coffee were invaluable to my research.

I would also like to acknowledge the financial support provided in part by Canada NSERC grant A5987 , Québec FCAR grant ER-0106, and my graduate scholarship from NSERC.

To Lisa and to my parents

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BIOS	- Basic Input/Output System
BRI	- Basic Rate Interface
CCITT	- International Telegraph & Telephone Consultative Committee
COSI	- Centralized Operating System Interfaces
CPU	- Central Processing Unit
CS	- Code Segment
EIOS	- Extended Input/Output System
EOB	- End Of Buffer
ES	- Expert System
HALOS	- High-Level Operating System Schedular
HARTS	- Hard Real-Time Support
HI	- Human Interface
H/W	- Hardware
I/O	- Input/Output
ICU	- Interactive Configuration Utility
ID	- Identifier
IDT	- Interrupt Descriptor Table
IP	- Instruction Pointer
IRR	- Interrupt Request Register
ISDN	- Integrated Services Digital Network

ISR	- In-Service Register
KBS	- Knowledge-Based System
LAN	- Local Area Network
MBX	- Mailbox
NT	- Network Terminating Equipment
OS	- Operating System
OSI	- Open Systems Interconnection
PC	- Personal Computer
PDLC	- Parallel Data Link Control
PIC	- Programmable Interrupt Controller (8259A)
PPI	- Programmable Peripheral Interface (8255A)
RAM	- Random Access Memory
ROM	- Read Only Memory
S/W	- Software
TE	- Terminal Equipment
UDI	- Universal Development Interface

CHAPTER 1

Introduction

The integration of computers and communications is a result of the shift from the so-called *commercial-industrial* society to the *information* society [1]. Today, there are approximately 600 million telecommunication subscribers worldwide [2]. Data communication currently accounts for 10% of network traffic in North America and is increasing at the rate of 30% a year [2]. To meet this growing demand for data communications, a unified, global digital telecommunications network known as the **Integrated Services Digital Network** (ISDN) has evolved [3].

Modern communications needs no longer are confined to voice services, rather, they now include a wide variety of data requirements. ISDN integrates voice and data services through one digital communication facility [3]. Users' communications needs do not only extend outside the boundaries of their homes but also within. There is an increased demand for the ability to communicate with electronic equipment within the home so that control of these devices can be possible [4]. A new type of system is needed that will support the user's computer and communication needs. In this thesis, we propose a new operating system architecture for such a platform. Collectively, we call this an **ISDN personal workstation.**

ISDN is currently in an initial implementation stage. Consequently, only modest uses have been made of the ISDN network. To exercise the current voice and data services, ISDN terminal designs normally comprise a personal computer, an ISDN access card, and a telephone [5]. These ISDN terminals satisfy current use of ISDN capabilities but as future applications evolve, we feel that terminals will no longer be adequate.

The purpose of our ISDN personal workstation is to overcome the shortcomings of current ISDN terminals. With the increased popularity of information services, future ISDN applications will be very information intensive. This will require decision making support for user applications. These types of applications could not easily run in current ISDN terminals with conventional operating systems. This stems from the fact that the architecture of conventional operating systems does not provide the framework needed for these information-based applications.

Thus, the design of our ISDN personal workstation is motivated by the following two fundamental features that users will expect from the workstation:

1. the ability to exploit ISDN information services

2. the ability to support autonomous control of the local environment in an intelligent home of the future.[1]

Let us now examine how these two motivations influence the design architecture.

ISDN provides the workstation with a gateway to both information and communication services, from which stem our strongest motivations. Widespread access to these services is made possible by the fact that ISDN will have service access points based on the **Open Systems Interconnection** (OSI) reference model[2]. With common protocols internationally standardized, users of ISDN can expect access to a network of various

[1] The Consumer Electronics Bus standard (CEBus) and the Smart House standard define communication protocols for the residential market in the United States [4,6].

[2] The International Telegraph & Telephone Consultative Committee (CCITT) have defined standards for ISDN implementation based on the OSI reference model. These standards include definition of protocols, services and network functions [7,8,9].

commercial third-party services. One very important type of service will be information retrieval applications using huge databases. An example illustrating this kind of service will be given in section 1.1.

Then why is the feature, of exploiting ISDN information services, such an important consideration in the design of the workstation? Certainly its usefulness to the user is obvious. The answer lies in the fact that this feature **necessitates special processing requirements**. A problem-solving, information-based application running on a workstation will have access to copious amounts of information *via* the ISDN. However, when the amount of knowledge/data to be processed is very large, heuristics must be applied to define a restricted set of knowledge/data to use in the problem solving [10,11]. In order for the workstation to effectively run information-based applications that use heuristic processing techniques, the *operating system must incorporate a software architecture that deals with abstract information.*

The second motivating feature in the design of the workstation is to provide the user with primitives for autonomous control. A computer that supports autonomous control can use its local environment without the user's direct intervention. The local environment defines the physical boundaries which the workstation can access and/or exert control over. We identify two types of autonomous control that the workstation should provide the user:

1. *remote digital control* via *ISDN*
2. *autonomous multitasking.*

Since ISDN provides the workstation with communication services, the potential for **remote control** exists. ISDN creates this potential because it can provide an economical and easily established digital channel[1]. Two computers can use the channel to exchange the command and telemetry information needed for remote control. A user would take advantage of this ISDN-based remote control by accessing home status information and initiating control functions from another station. Clearly, the architecture of the workstation must include facilities for centralized control of the interconnected devices in its local environment to achieve autonomous remote control.

Autonomous multitasking, the second type of autonomous control, is the name we use to define a special class of operating system services. It is a conglomeration of capabilities that permits control applications to execute on their own. These control applications would normally require some user intervention if they were running on personal computers with a regular multitasking operating system. However on this workstation, operating system support permits completely unattended operation. For instance, the user could run a workstation application that transfers a large file across the ISDN late at night. In section 1.1, this capability is highlighted with an example of a problem-solving application that gathers large amounts of information through the ISDN without disturbing the operator.

To support autonomous multitasking services, the workstation requires more than just a centralization of control devices. Rather, provisions are needed in the software

[1] The CCITT defines the D-channel in ISDN basic rate service as a 16 kbit/s, packet switched, common access channel. The D-channel is intended to carry signalling information for circuit-switched channels and telemetry [12].

architecture for a set of high-level primitives to exploit these devices. When teamed-up with a set of real-time primitives, this architecture gives workstation users a platform capable of concurrent, intelligent problem solving.

By examining the motivations behind the design of the workstation, we have identified two key requirements for its software architecture. One requirement was **architectural support for heuristic processing** of the information available through ISDN third-party services. The second was a provision in the **software architecture for managing centralized control** in order to have an autonomous control facility. A software architecture built around a **knowledge-based system** (KBS) provides a solid foundation to achieve these goals.

1.1 Example Operation

The purpose of this example is to illustrate how we envision the workstation will be used and to emphasize its architectural requirements. This example will serve to clarify subsequent discussions of operating system requirements.

We have selected a real estate application to demonstrate the practicality and strength of a knowledge-based system being used in a non-trivial, knowledge/data-intensive situation. The ISDN personal workstation makes it possible to support KBS applications with its communication, display, computing and control facilities. Features of the workstation we wish to highlight in this example include: rule entry, autonomous multitasking and access to external data bases *via* ISDN.

(futuristic office scenario)

You have been advised that your department is being relocated to a new city. Lacking the time to personally look for a new home, you make use of a real estate expert system application which is available on your new ISDN personal workstation. After loading the software into the computer, you begin your search.

The expert system application (provided by the local real estate company) comes equipped with the knowledge of how to access all the necessary real estate information. Using ISDN, the data on these external data bases is readily accessible. Before the search can begin, you must use the workstation's knowledge-acquisition utility, to enter new rules. These rules will augment the existing selection criteria to reflect your own specifications.

After entering a trial set of rules, you run the expert system application for a short time to obtain some feedback. In the meantime, the autonomous multitasking operating system frees you to work on any other application. Once a few candidate houses have been identified, you are notified of the progress. You can now verify the rule set with the workstation's explanation utility which elucidates the application's decision process. In addition, you can visually examine the houses using graphic images of its exterior and interior displayed on the workstation's high resolution monitor. Also, relevant information such as property value, taxes and utility expenses can be reviewed. After this review, you may want to modify or add rules to the set to improve the selection process.

Satisfied that all relevant information has been included in your personal rule-set, you permit the expert system application to run autonomously until it is ready to present the best available candidates. In the meantime, you begin another application.

1.2 Research Contributions

The purpose of my research was to propose a knowledge-based, event-driven, real-time operating system architecture for an ISDN personal workstation. The design of the operating system architecture has been motivated by the features that were discussed in the introduction. The architecture differs from conventional personal computer operating systems because it supports a knowledge-based system and the capability of autonomous control. In developing this new operating system architecture, the following relevant contributions were made:

1. We extended the basic error detection and reporting capabilities of traditional operating system device independent I/O software [13]. In our proposed operating system, we defined an **autonomous exception handling** capability that allows error-tolerant autonomous control applications to run.

2. We broadened the concept of device independent I/O [13] to include all operating system services. This provided the necessary **interfacing support** needed by the knowledge-based system.

3. We provided a soft real-time, event-driven environment for applications by forming a **hierarchical scheduling organization**. Through this scheme, we were able to add a high-level scheduler that would intelligently control the lower-level conventional

scheduling mechanism. As a result, our operating system can provide both multitasking and multi-inferencing support.

4. We applied the idea of symbolic processing, used in network management systems for fault isolation [14], to the workstation operating system. A layer in the operating system is used to perform the symbolic processing necessary for **event interpretation**. This processing is necessary to support the workstation's autonomous control capability.

5. We applied the **event-driven** structural organization used in network management and control systems [14,15] to the organization of the workstation operating system. We did this so that our workstation could support the autonomous control of its local environment. The main feature that an event-driven organization provides is the ability to deal with unpredicted changes in state of the local environment. This involves making events capable of starting new applications.

In the course of implementing our experimental ISDN personal workstation we developed a general purpose, bidirectional, high-speed, parallel data bus hardware and software module that provides a layer 2 communication link to the workstation. In addition, this module also provides a frame routing service for incoming messages so that multiple logical access can be provided. This module was used to provide a high-speed, **digital communication link** between two separate platforms.

1.3 Thesis Outline

Having identified the motivations behind the design of our ISDN personal workstation in the Introduction, Chapter 2 will discuss the architectural requirements of the operating system[1]. After this, Chapter 3 will present our operating system architecture in detail and demonstrate how the requirements have been met. In Chapter 4, we will present our experimental ISDN personal workstation architecture to demonstrate a possible implementation of the architecture discussed in the previous chapter. In Chapter 5, we summarize our research work, present our conclusions and suggest related aspects and topics for further studies.

[1] Reference [16] contains a discussion of the ISDN personal workstation's knowledge-based system requirements, architecture and experimental implementation.

CHAPTER 2

Software Architecture Requirements of the ISDN Personal Workstation

The features motivating the design of the ISDN personal workstation were identified in the previous chapter. We will now examine the software architecture requirements needed to deliver these features.

In Chapter 1, we showed that a software architecture built around a knowledge-based system is necessary to support our two fundamental design features. The first, exploiting ISDN information services, requires heuristic processing [17]. This requirement can be satisfied by the heuristic processing capability of a knowledge-based system [18]. The architecture of the ISDN personal workstation's KBS is covered in reference [16]. The requirement for the second feature, autonomous control, is covered in this chapter.

2.1 Workstation Interface Primitives

Workstation interface primitives are necessary in the operating system so that the local environment (*i.e.* the workstation itself as well as the devices connected to it) can be controlled. Control of the local environment is possible through either the user's direct intervention (user shell) or from an autonomous control application (a knowledge-based system). In either case, the same set of workstation interface primitives is required.

A common interface standard supports the idea of centralizing access to all workstation interfaces through one common access point in the operating system as shown in Figure 1. This structure facilitates the addition of third-party products to our system as well as the modification of low-level device interfaces. A reason why the workstation primitives should not be directly implemented in a KBS is because the operating system can better deal with the *procedural knowledge* necessary for controlling devices. The operating system will isolate the complexities of the device-level workstation interfaces and provide access through intelligent primitives. These primitives must be sufficiently abstract so that complex controlling commands can be easily specified.

Because workstation interfacing is centralized, the set of abstract primitives permits the definition of a single, *unified* workstation language that can be utilized

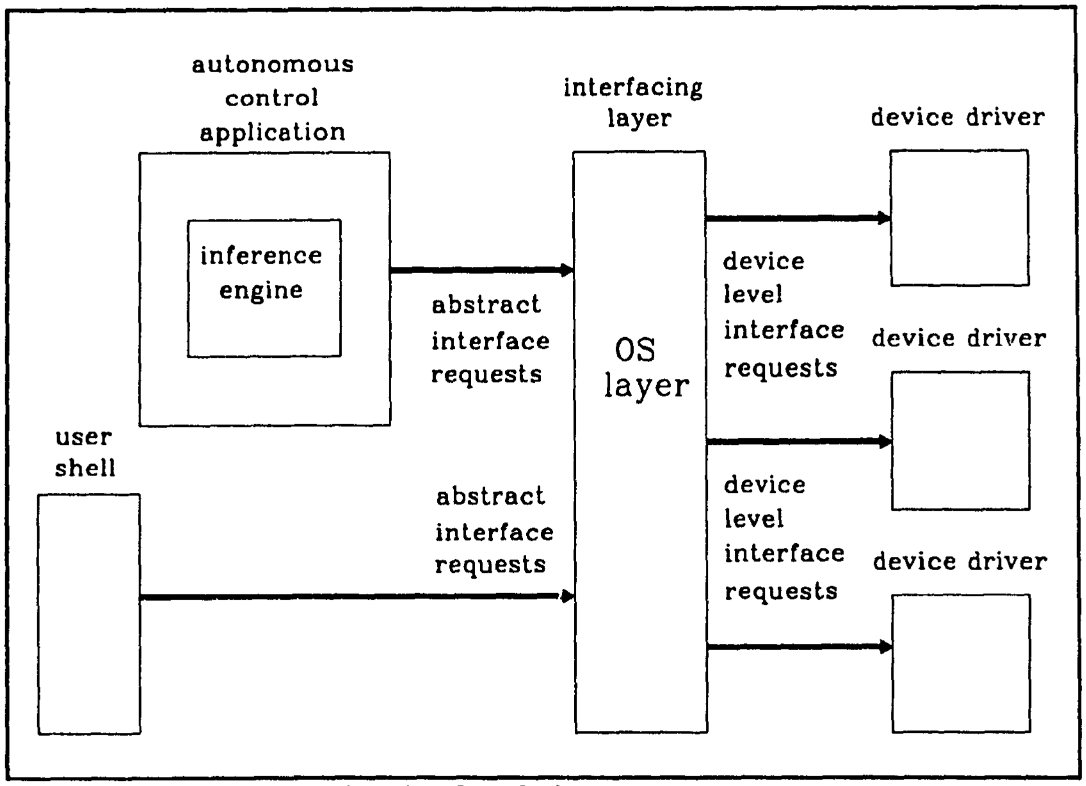

autonomous control application
inference engine
abstract interface requests
user shell
abstract interface requests
interfacing layer
OS layer
device level interface requests
device level interface requests
device driver
device driver
device driver

Figure 1 Centralized Workstation Interfacing.

throughout the entire workstation environment. The reason why we call the language unified is because it uses only one set of primitives. The workstation language has features similar to those found in interpretive BASIC where the same syntax is employed in system calls and applications [19]. However unlike BASIC, the workstation language can be easily used to perform complicated control functions because of the underlying set of abstract primitives.

2.2 Real-Time Systems

An important aspect of autonomous control is the ability to execute applications at a pre-specified time (real-time scheduling). Therefore, operating system support of timing constraints and ability to handle timing exceptions are crucial in our system. Before we specify the workstation's real-time requirements, we will first discuss the characteristics of real-time computing.

Real-time systems have been divided into two categories: *hard real-time* and *soft real-time* [20]. We make a further distinction between the two categories by stipulating that hard real-time systems are interrupt driven systems.

2.2.1 Hard Real-Time

Hard real-time systems support the critical processing requests made by hardware devices. The requests are made to the host computer *via* hardware interrupts. These requests demand prompt, uninterrupted service. The implied timing constraint associated with these requests is that processing should begin within a *reasonable* amount of time.

We will now determine what is a reasonable amount of time by examining **interrupt latency.**

Hard real-time operating systems are characterized by their maximum interrupt latency. In general, interrupt latency is the time that elapses between the instance a hardware device makes an interrupt request to the point that its interrupt service routine begins [21]. In other words, how long the device must wait before being served by the host computer.

A unique characteristic that distinguishes hard real-time operating systems from conventional (non-real-time) operating systems is that their interrupt latency is bounded. The reason why this delay must have an upper limit is because most devices can only wait a short amount of time for service. In a hard real-time system, the response time is viewed as a crucial part of the correctness of the software [22]. For such systems, if service begins late, then frequently the computational results will be incorrect. For example, consider an ISDN access device receiving data into a fixed size buffer at a continuous rate of 64 kbit/s. Once the buffer is full, the device notifies the host computer. If the host computer's response time is greater than the transmission rate, then the data in the buffer becomes corrupted. An operating system supporting such hardware, must guarantee that its timing constraints will always be met. The only way an operating system can *start* to make such a commitment is if its interrupt latency is bounded.

Interrupt latency consists of the time it takes the interrupt support hardware to process the device request and the time required for the computer to accept the interrupt request and start the interrupt service routine. There is one other critical factor which affects interrupt latency and distinguishes a hard real-time operating system from a conventional operating system. This factor is the contiguous length of time that interrupts are disabled by an operating system's kernel during system calls.

To illustrate this point, a graph of the interrupt latency as a function of the maximum amount of time that interrupts are disabled by an operating system is shown in Figure 2. This graph shows that after accounting for the unavoidable hardware related delay in servicing interrupts, (the step at t_0), the interrupt latency becomes a linear function of the time that interrupts are disabled. In a non-real-time operating system, this could be of unbounded duration (shown as dashed line). However in a hard real-time operating system, the kernel is designed with the constraint that, during system calls, it cannot disable interrupts for a contiguous length of time that exceeds t_r. This feature guarantees that there will be an upper limit on latency (T_l).

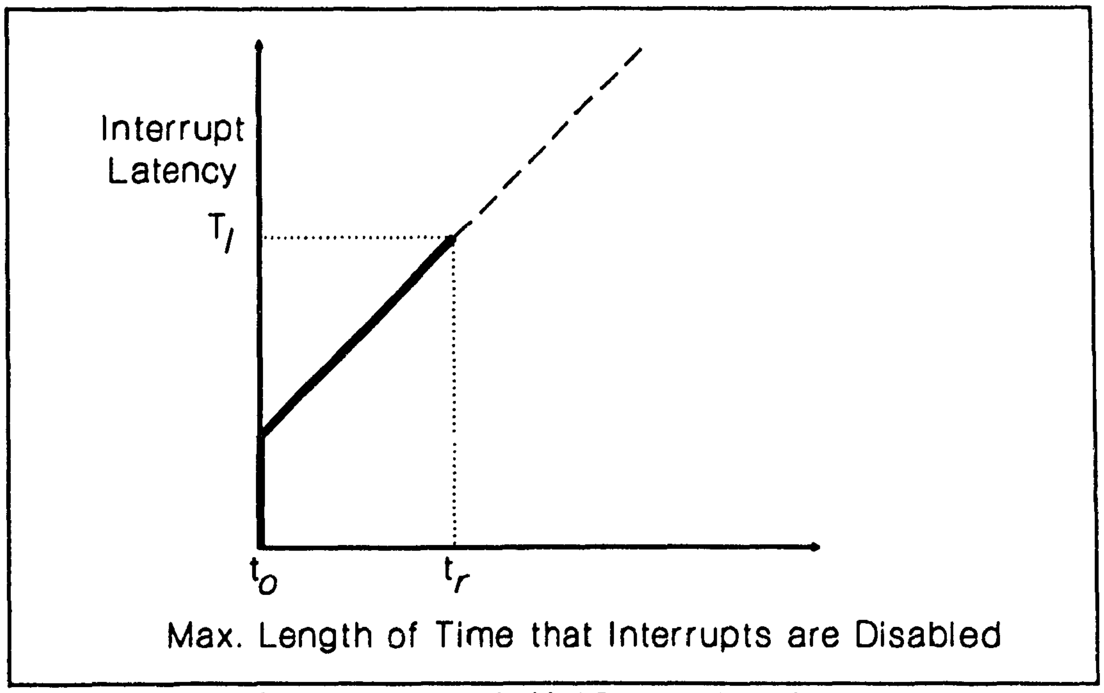


Figure 2 Interrupt Latency Versus Disabled Interrupt Duration.

There is a common misconception that swift average response time makes a computer system a hard real-time system [23]. The incorrectness of this misconception can be easily seen in the graphs of Figures 3 and 4. In these graphs, the probability density functions (PDFs) of the interrupt latency are shown. For the system with fast response time and an ordinary operating system (Figure 3), the graph shows that, on average, the interrupt latency is small --just like the hard real-time system (Figure 4). However, there is a non-zero probability of having a large interrupt latency that would result in a long response time. This possibility makes it a non-real-time system.

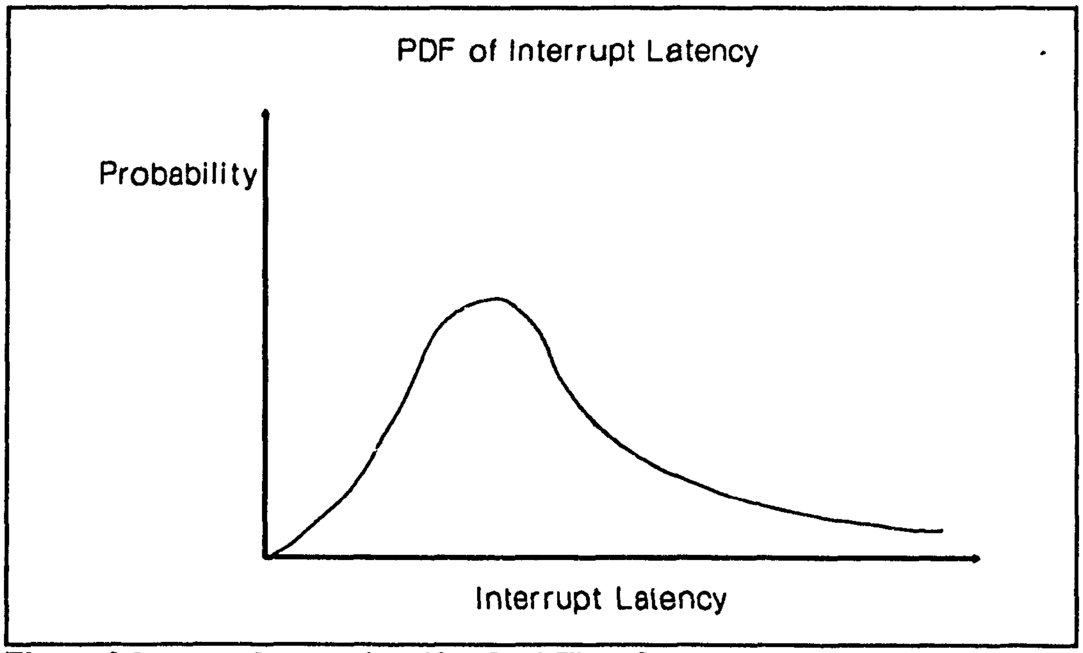


Figure 3 Interrupt Latency in a Non-Real-Time System.

So what kind of devices can be supported by a hard real-time operating system? Any real-time device can, as long as the maximum interrupt latency of the operating system is acceptable to it (*i.e.* our *reasonable* amount of time). If this maximum delay is not longer than the device's timing deadline, then the device is guaranteed proper response time from the operating system. Conversely, if the maximum interrupt latency of the operating system exceeds the response time demanded by a hardware device, the implicit timing constraints of the device are not met.

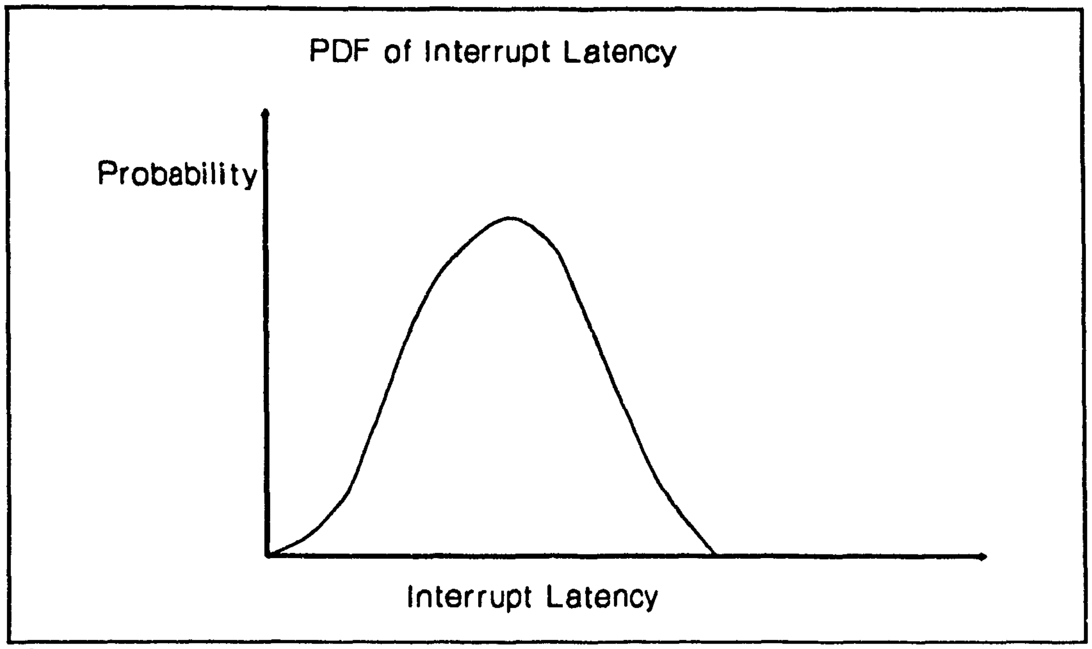


Figure 4 Interrupt Latency in a Hard Real-Time System.

2.2.2 Soft Real-Time

Soft real-time systems support the non-critical, time dependent processing requirements of real-time software applications. In addition to being logically correct, a real-time program must have its timing constraints satisfied [24]. There are two types of real-time processes: periodic and sporadic [20]. A periodic process needs to be scheduled (*i.e.* becomes ready) at regular intervals while a sporadic process may become ready at any time [24].

A real-time process explicitly specifies its processing requirements in terms of actual timing parameters (timing constraints). The timing constraints can be either absolute (time-of-day) or relative (deadline specified with respect to the time a process starts) [24]. A soft real-time operating system must endeavour to schedule these processes so that they can meet their timing constraints. However, unlike in a hard real-time system, the operating system cannot guarantee that it will meet all timing deadlines. Consequently, failure to meet timing constraints results only in a non-critical error condition called a **timing violation**.

Timing violations are dealt with by the operating system through the use of exception handlers. The operating system must be able to detect a missed deadline and then schedule an exception handler to initiate a recovery action. Possible actions include: notifying the user about the problem and either halting the process or rescheduling the process using new timing constraints defined in the exception handler [24]. The exact recovery plan will depend upon the nature of the application.

Soft real-time operating systems are characterized by their time-based process schedulers. Various scheduling strategies for a uni-processor preemptive scheduling environment have been defined [20,22,25,26,27]. In general, the scheduler does not only attempt to meet the timing constraints imposed by software, it must also try to maximize utilization of the CPU [28]. Consequently, not all of the timing constraints in the system can be met. Performance evaluation criteria for schedulers typically include the number of missed deadlines, the degree of processor utilization and scheduling stability [27].

In order to formulate a scheduling strategy, the scheduler requires initial timing parameters from the software it runs [26]. In general, the timing parameters include: the acceptable maximum and minimum amount of delay time before starting execution, the estimated maximum execution time of a computation, and the latest time in which execution should be completed [24]. In order for the scheduler to receive these parameters, the programmer needs a mechanism to communicate them to the scheduler.

Since the ISDN personal workstation must support both hard and soft real-time scheduling, its architecture incorporates the real-time characteristics outlined in this section.

2.3 Scheduling Control

A consequence of autonomous control is the need to support directly connected external devices. Some of these devices may have hard real-time processing demands. The devices with such demands would need real-time device drivers. To meet these demands, the workstation requires an operating system kernel with a bounded interrupt latency that is small enough to satisfy the real-time devices. Built on such a kernel, the real-time device drivers would be capable of supporting these devices.

The soft real-time requirements stem from the workstation's autonomous control feature. A mechanism for specifying timing constraints that control application execution are needed. In addition, a means of dealing with timing violations is also essential.

Another aspect of autonomous control is the ability to execute multiple concurrent applications autonomously. The operating system must provide a mechanism to assign priorities to applications to deal with a workstation which may run more applications than available processors. The operating system's dynamic prioritizing mechanism must take into account several factors before forming a scheduling priority. A main factor should be the user's specified importance of the application so that it will receive an appropriate amount of service time.

In the next chapter, these workstation operating system requirements will be addressed and an architecture will be proposed.

CHAPTER 3

Structure of an Event-Driven, Real-Time Operating System

The desire to have a workstation that can autonomously control its local environment led us to use an **event-driven** operating system architecture. This design approach has been used effectively in network management and control systems [14,15]. In these systems, events are viewed as a problem-related change in state of their communication network. Network events result in the generation of alarm messages that signal the presence of a problem. These alarms are then processed by a control center to determine causal and temporal relationships and severity [15]. After this, the error problem becomes well defined and undergoes system-initiated diagnostic processing. Thus all processing initiation depends on the arrival of events, hence the term event-driven.

Similarly, the notion of an event is essential in the design of our own operating system. We broadly define an **operating system event** (OS event) as the ***awareness that an unpredicted change in state of the local environment has occurred.*** Examples of OS events are commands issued by the user, the arrival of an incoming ISDN call or a signal from an alarm system connected to the workstation. In each of these cases, the operating system could not know in advance when these incidents would occur.

Not all external activity is considered to be OS events. When it is under direct software control, external activity does not represent an unknown state change. For example, we would not consider the consequences of an application using the ISDN connection (*i.e.* change in channel utilization) to be an OS event. The software would certainly know the consequences, and any hardware activity would be predictable.

For the operating system to be event-driven, it must be able to detect unpredicted changes in state of the local environment and use this information to start applications. To do this, the operating system must rely on interrupts to notify it of any such changes. As a result, the operating system must have the ability to process interrupts to the point of understanding their implied processing requirements. Subsequently, it must also be able to act on such requests.

Our operating system must also support the autonomous control requirements imposed by KBS applications. For these needs, the operating system provides an abstract set of primitives which the applications may use. In fact, this operating system requirement contends with the need for it to be event driven.

3.1 The Layered Organization of the Operating System Architecture

The requirement that our workstation supports both OS events as well as KBS application processing has led to a single operating system architecture with a dual set of primitives[1]. The combined, hierarchical organization of our operating system into a layered structure is depicted in Figure 5. The operating system has two distinct sections corresponding to OS event and application processing functions.

The layers situated in the upper half of the operating system diagram, shown in Figure 5, have primitives that provide services to applications and thus these layers are

[1] In an operating system context, primitives are defined as abstract services supported by the operating system [29].

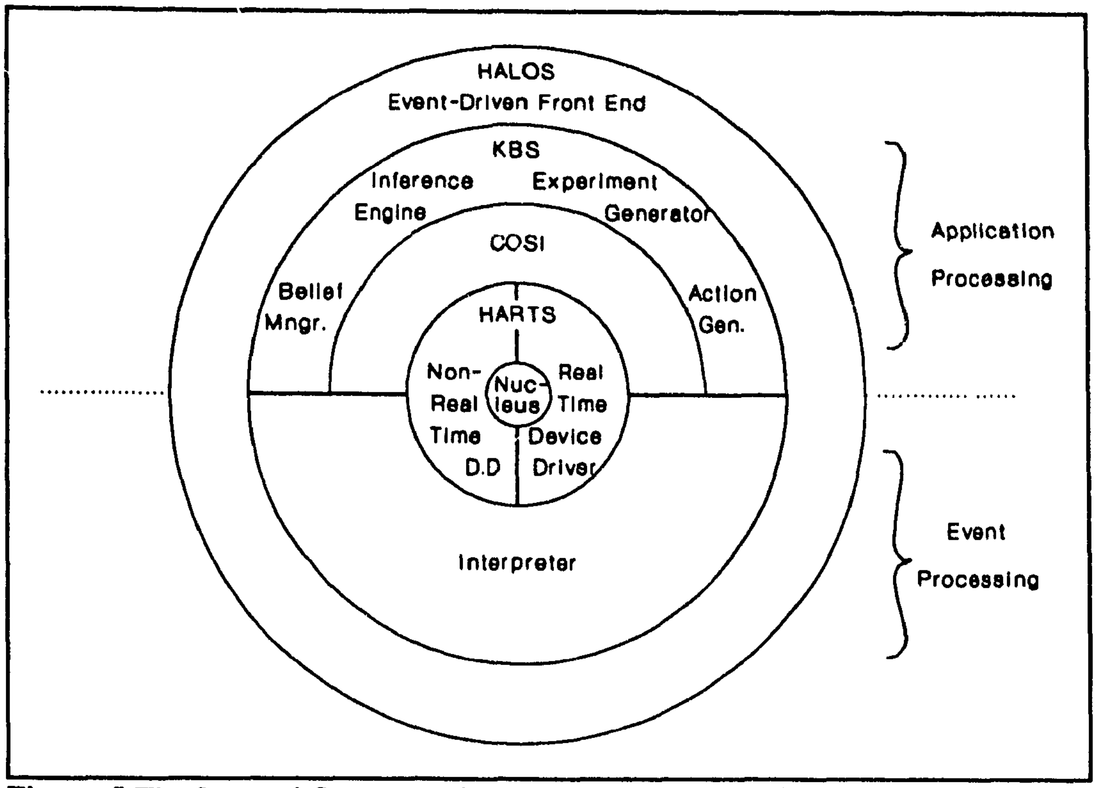


Figure 5 The Layered Structure of the Workstation Operating System.

called the application processing section. Access to these services is the same as in conventional operating systems in that applications depend on the support offered in lower layers of the hierarchy.

The application processing section is organized into five functional layers. The top layer is an event-driven schedular called the High-Level Operating System Schedular (**HALOS**). The second layer corresponds to the Knowledge-Based System (**KBS**) architecture that supports expert system applications [16]. Services in the third layer provide a centralized access to all workstation interfaces. This layer is called the Centralized Operating System Interface (**COSI**) layer. The fourth layer includes the device drivers needed for both real-time and non-real-time hardware support. This layer is called the Hard Real-Time Support (**HARTS**) layer. At the core of this diagram is the hard real-time, multitasking KERNEL layer.

The procedures in the lower half of the diagram shown in Figure 5, preprocess OS events and hence are called the event processing section. OS events must be preprocessed because, initially, the operating system has no knowledge concerning their meaning. From the operating system's point of view, OS events are not correlated with any previous activity. Before they can be acted upon, they need to be interpreted.

The difference between the event processing and the application processing sections is that layers two and three are now replaced by a single event interpreter layer. The event interpreter determines the meaning of incoming requests and passes this information up to the event-driven front end.

3.2 OS Event Processing

Previously, we defined an OS event as "the awareness" of an external activity. We now extend this definition to include the movement of this knowledge through the event processing section of the operating system. In this context, an OS event proceeds through three states (Figure 6). We say that the duration of an event is bounded by the interrupt that starts it , and the point where the operating system has mapped it into a well defined request.

We define a *request* as the information that a hardware device provides the operating system in order to get an application started. For example, if a home monitoring system detected an intruder, then an interrupt signal along with other information (like sensor identification and status codes) would be sent to the workstation. The information

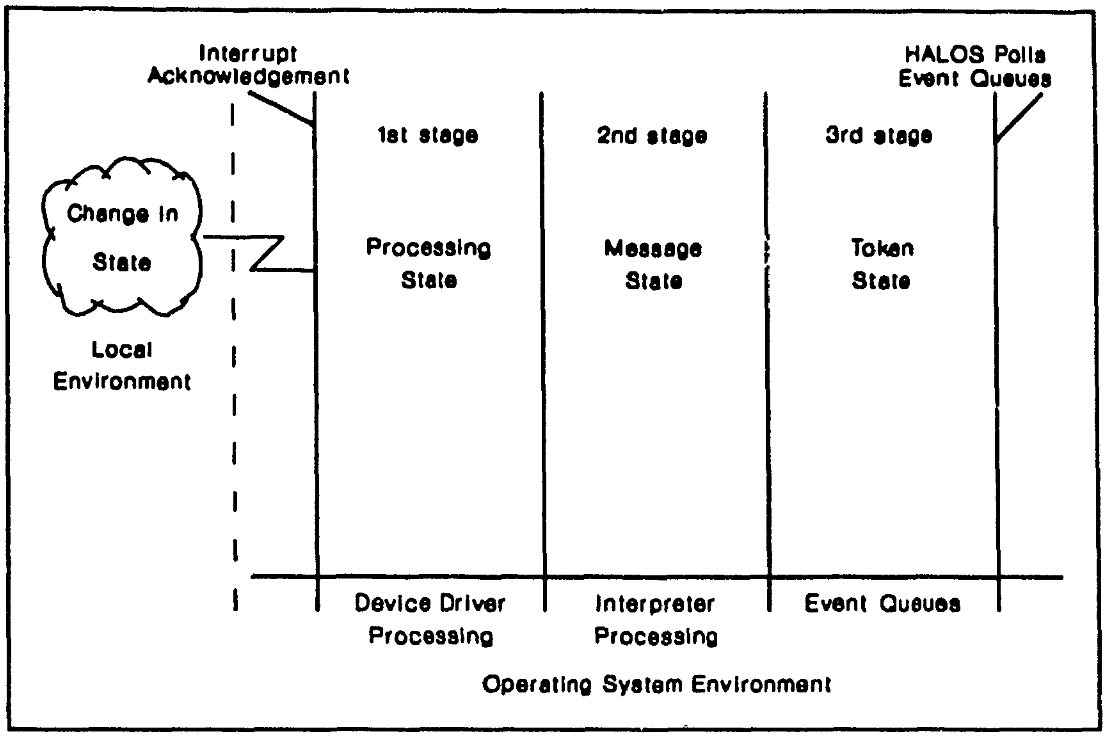


Figure 6 Stages of an OS Event.

would then be processed to identify the request (*e.g*. differentiate a burglar alarm from a fire alarm). The identified request would then result in the scheduling of an application. In the event of a burglar alarm, the application could then make an ISDN phone call to the police. To see how our operating system maps OS events into a request, we will examine in depth the three states of an OS event.

The flow of OS events begins with the acknowledgement of an interrupt request by the operating system. The first stage of an OS event is device driver processing, where the immediate real-time demands of the hardware are met. After this initial processing (*e.g*. receiving data into a buffer), the OS event becomes a message (stage 2) which the **interpreter** processes as shown in part b of Figure 7.

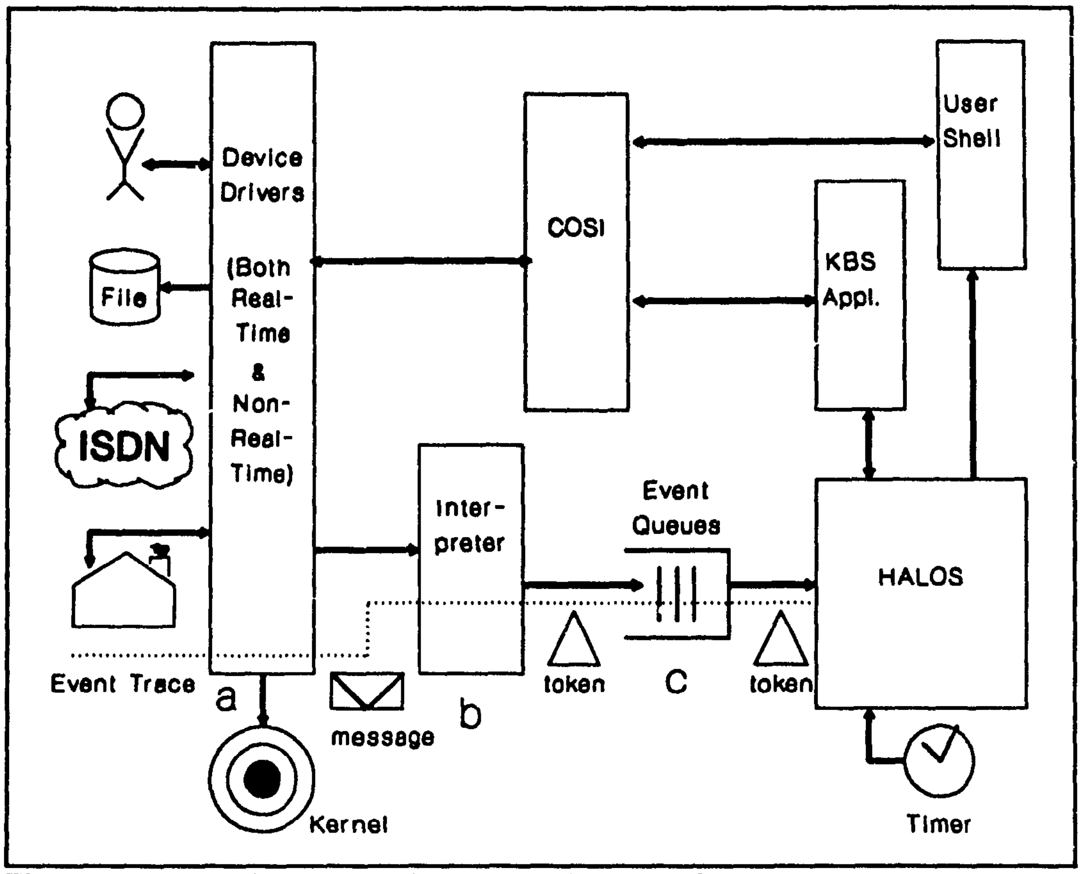


Figure 7 Flow of OS Events Through the Operating System.

Stage 2, or the message state, is analogous to the alarm messages that arrive in network management and control systems [15]. The processing that the interpreter performs is to recognize the message and translate it into a standardized format we call a token[1]. This processing stage is comparable to a network management system that determines the relationships and severity of alarm messages. In our system, the token is then placed into an appropriate **event** queue as shown in part c of Figure 7, at which time the event enters the third and final stage.

[1] We define a token as a symbolic representation for a particular service supported by the workstation's operating system.

The last stage ends when HALOS retrieves the token. At this point, the notification of an unpredicted change in state has been transformed into a well defined application *request* which is ready to be executed. This is similar to the diagnostic state of a network management system [15]. At this point, HALOS prioritizes the complete set of new requests and is ready to start an application.

Those applications started by HALOS can make use of the COSI layer of the operating system. COSI provides an intelligent, standardized interface so that applications interact with the workstation interfaces at an abstract rather than the more complicated device level. We will next examine in detail each layer of the operating system architecture.

3.3 The Hard Real-Time Support (HARTS) Layer

As in all operating systems, our architecture has a set of programs, typically interrupt driven, to control hardware devices. These programs are usually called device drivers and are organized in a layer above the kernel. But unlike most other systems, our workstation supports hardware devices with hard real-time requirements[1]. Therefore, our device driver layer has been designed with the needs of hard real-time in mind.

The device driver layer of our operating system is entitled **HARTS**. Both hard real-time and non-real-time device drivers make up this layer. All device drivers in HARTS are divided into three independent routines. In order for these routines to provide hard real-time support, the underlying kernel must guarantee a maximum interrupt latency.

[1] *Hard real-time* has been defined in section 2.2.1.

3.3.1 Device Drivers

A device driver consists of three separate routines shown in Figure 8. Each routine provides a different degree of real-time processing. This division in processing is due to the various timing constraints that hard real-time devices have, which can be ranked into: *critical*, *immediate*, and *ordinary* classes.

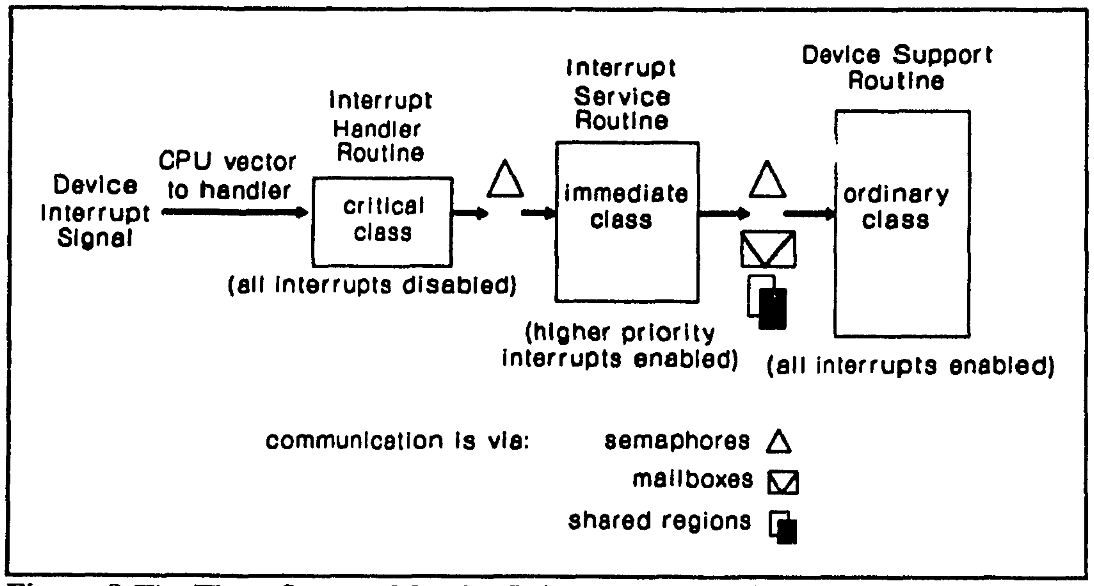


Figure 8 The Three Stages of Device Driver Processing in HARTS.

Any device with a *critical* timing constraint, requires that its processing begins as soon as its interrupt signal becomes acknowledged. The amount of acceptable delay is limited to the interrupt latency of the kernel. Critical processing must not be interrupted, therefore all interrupts need to be disabled during this time. The routine that does this class of processing is called the *interrupt handler* routine of the device driver. Since all interrupts are disabled in an interrupt handler, the amount of time spent in these routines must be minimized so that other high-priority devices can get serviced in time.

A processing classification with a more lenient timing constraint is the *immediate* class. For this class, the permissible delay in completing processing is larger. The completion delay includes the time that might be spent waiting if preempted by a higher priority device. The *interrupt service* routine of the device driver is responsible for this processing. An interrupt service routine may be preempted by an interrupt from a higher priority device since, at this stage, higher priority interrupts are enabled. These interrupts are enabled during an interrupt service routine so that the operating system may satisfy any new critical processing requirements.

The final ranking of timing constraints defines a class that permits an even larger overall service time. This class of processing, called *ordinary* class, is performed by the *device support* routine of a device driver. During ordinary class processing, all interrupts are enabled so that any device with a new processing request can be serviced by preempting the device support routine. Even a device havir g a lower priority can preempt a device with a higher priority during ordinary processing.

Hard real-time devices can have timing constraints in each of the three classes. For these requests to be processed efficiently, all three classes must be supported by our operating system. For instance, if ordinary constraints were processed by a higher ranking class, such as an interrupt handler, then interrupts would needlessly be disabled. As a result, during this processing time, no new device requests could be serviced. This inefficient implementation would increase the chances of violating a critical timing constraint in a pending device request, thus decreasing the number of real-time devices the operating system could support.

Device driver processing logically progresses through each of the three routines. The amount of processing done at each stage depends on the hard real-time requirements of the device. If there is no processing required at a particular routine, then the next routine is started immediately. At the end of a device support routine, if the device interrupt signified the start of an OS event then an interpreter stage would begin.

3.3.2 Dedicated Processors

Another consideration in the HARTS architecture is the use of *dedicated processors* to support certain devices. The dedicated processor approach involves establishing a processor and local memory on a separate board, exclusively for a single device. We specify three instances where the type of processing required by a device is better served by the dedicated processor approach rather than having the workstation do all of the processing.

The first instance is when the volume of processing required by the device is so high that it would degrade the performance of knowledge-based applications. The second instance occurs if the hard real-time support (maximum latency) of the workstation is insufficient for a device. The last instance arises if the nature of the device processing is better suited for a special-purpose processor. For example, using a signal-processor for video, or a micro-controller for ISDN communication involves a special-purpose processor.

We defined a general protocol for the interface between dedicated processors and HARTS. The dedicated processors should reside inside the workstation, connected to its

I/O backplane. The workstation, therefore, requires an I/O backplane that can support such boards. The boards should have their own operating system kernel that can communicate with the workstation operating system through a high-speed interface. To minimize the processing load this would present to the system, the interface involves setting up data in some type of dual access memory (*e.g.* Dual-port RAM), and then signalling the workstation to rapidly read it using direct memory access (DMA).

The dual access memory protocol for dedicated processors allows them to move data directly into the workstation. In all other aspect, these boards will interface to HARTS as standard devices that use interrupts to signal a HARTS driver. This driver will perform any necessary *ordinary class* hard real-time processing which may lead to the start of an OS event.

3.4 The Interpreter Layer

In order for our operating system to support event processing, it must be able to process OS events to the point of understanding their implied application requests. The layer of the operating system which performs the symbolic processing [10] required to determine the meaning of a device interrupt is called the *interpreter* layer. This type of symbolic processing is similar to that used in network management systems for fault isolation [14]. In network systems, the meaning of alarm messages must be determined so that a program can be run which finds and corrects the problem-causing component.

Device drivers obtain preliminary information about an OS event during HARTS layer processing. This information is then forwarded to interpreters as was shown in

Figure 6. Hence, the last two steps of device driver servicing of OS events involves placing any new available information into a shared memory location and then signalling an interpreter. Upon receiving a device driver signal, an interpreter knows that it has an OS event message and retrieves it.

In the workstation operating system, each source of an OS event (user, ISDN, centralized controller) has an associated specialized interpreter. Interpreters process the information (message) obtained in the previous device driver stage. The messages represent predefined operating system commands. A command would typically be a request to start a particular application.

The interpreter parses each message to extract the information needed to understand the command. Next, using a look-up table containing a list of all possible commands, the message is translated into a standardized format called a **token**. The token identifies, for the operating system, what application should be run by the front-end as a result of the OS event.

Messages from the user are translated into commands by a natural language processor (user interpreter). Messages from devices, such as an ISDN access card, are processed by a device (ISDN) interpreter. For example, an incoming ISDN call provides the ISDN interpreter with a message containing information imbedded in a packet. This message includes such things as line activity (ringing), channel identification, caller identification, *etc.*

Interpreter processing is completed when the token is placed into the event queue as;ociated with the interpreter. In this example, the token might specify that an ISDN voice call application should be started. The event queues serve as the interface between the interpreters and HALOS. HALOS does not wait for any signal from the interpreters to retrieve a token from an event queue instead, it periodically checks the queues.

3.5 The High-Level Operating System Schedular (HALOS) Layer

In all multitasking operating systems, the amount of service that users receive is decided by its scheduling mechanism. Normally, a scheduling mechanism only works at the level of individual processes. Its job is to decide which processes should run next by adhering to either a priority or a first come first serve algorithm. A distinguishing feature in our operating system is that we add a further layer of scheduling above the conventional scheduling mechanism.

The addition of a high-level schedular, called HALOS, into the ISDN workstation architecture forms a scheduling hierarchy. Figure 9 illustrates this hierarchy. Conventional scheduling of processes and *KBS problems*[1] is performed by the kernel and inference engine respectively. Situated on top of the hierarchy, HALOS controls the scheduling of all *applications*[2] by manipulating the conventional schedulers.

The kernel of any multitasking operating system is composed of routines that provide memory and process management to the rest of the system. The key component

[1] A problem is an instance of a heuristic computation running on the inference engine of the KBS. For a detailed discussion on how the workstation's KBS treats problems, see reference [16].

[2] An application request is a general computational request for either a KBS or a non-KBS program.

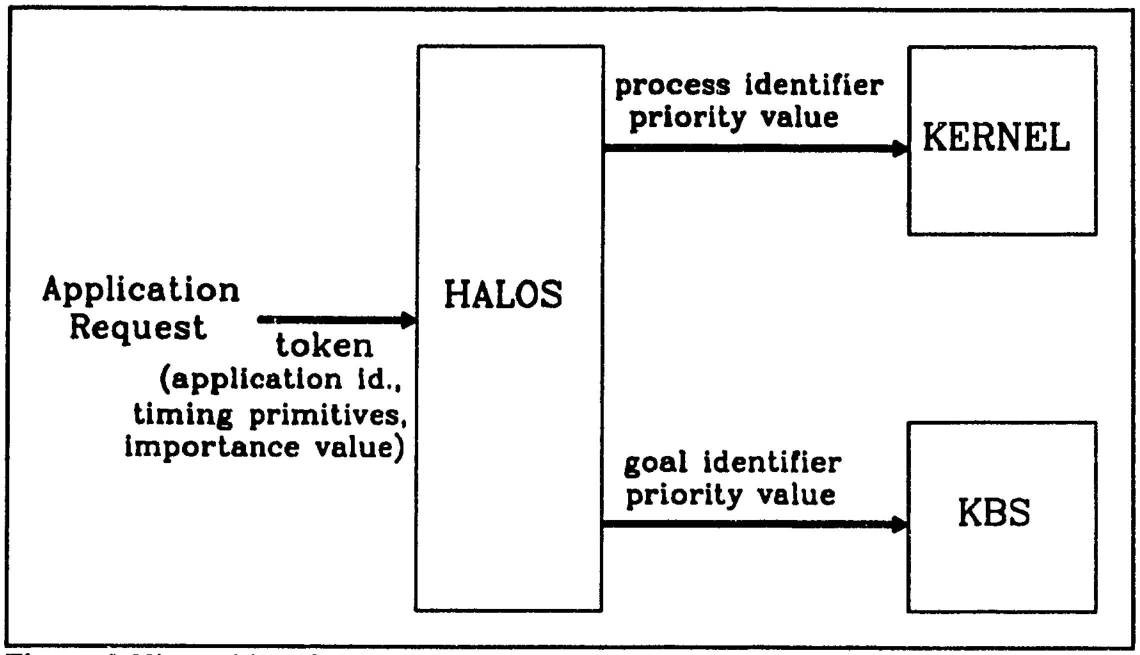


Figure 9 Hierarchical Scheduling used by HALOS.

of process management, in such kernels, is the process schedular. A process schedular maintains the illusion of concurrent processing by rapidly switching one processor among several processes [29]. The decision of when to switch, and which process to schedule next, depends on process priorities and the scheduling algorithm. Also, the process schedular ensures that the software context is saved during a context switch so that a process can be continued again at a later time.

In our operating system, the kernel has such a priority-based, preemptive process schedular. The kernel schedules processes according to their assigned priorities using the round-robin algorithm [13]. Preemption is required so that newly arrived, higher priority requests do not have to wait until a lower priority process blocks. Preemption is normally implemented by giving running processes a time quota. When the time quota expires, the running process is exchanged with another ready process.

In addition to the kernel, there is also a schedular for the multi-inferencing inference engine. Instead of scheduling processes, this schedular schedules *problems* for the inference engine. It is also priority-based and preemptive. The architectures of the inference engine and its schedular are given in reference [16].

As we indicated, our application schedular controls both the process and the problem schedulers. The application schedular receives application requests and decides the order in which each application should start executing. The schedular assigns a *dynamic priority* to the requests, as will be explained in section 3.5.1. After this, either the process schedular or the problem schedular manages the actual execution of the individual applications.

The reason for using a hierarchical scheduling organization is to provide a soft real-time[1] event-driven environment for applications. The mechanism by which HALOS achieves this will be presented next.

3.5.1 Prioritizing Information

Since both the kernel and inference engine have priority-based schedulers, the means by which HALOS can exert some control is by assigning a dynamic priority to application requests. HALOS' prioritizing function takes into account two types of information: predetermined data and dynamic operating system state information.

Predetermined data is defined by the application programmer and consists of three specifications. The first specification is the importance of the application (expressed

[1] *Soft real-time* has been defined in section 2.2.2.

numerically) relative to other applications. Th:ough this data, the programmer communicates how each application is ranked.

The second specification consists of soft real-time parameters (timing primitives). The timing primitives define constraints such as: the maximum and minimum acceptable time before starting an application, an approximation of the maximum execution time of an application, and the latest allowable completion time. By supporting these timing primitives, HALOS provides an environment which facilitates the writing of soft real-time applications.

Finally, the last specification depends on where the application will run. For KBS applications, the name of the starting goal[1] is expressed. For other applications, a process identifier is given. This specification is stored in the header of every application file.

The dynamic operating system state information consists of two values. The first value is the time-of-day. The second value defines the system load. This value indicates the number of processes in each priority level for all the possible states (running, ready, blocked, sleeping, suspended).

The dynamic operating system state information is used by HALOS to calculate a new application's *dynamic priority*. The current load condition of the operating system must be known in order to accurately p:edict application performance. A high-level schedular must know this because, only through a proper knowledge of performance can such programmer specifications as importance and timing primitives be met.

[1] The initial goal is the first rule processed by the inference engine, and is used to start KBS applications [16].

HALOS calculates a dynamic priority, using current state information, to ensure that each application will receive a sufficient amount of processing to meet its soft real-time demands. The calculated dynamic priority also reflects the relative importance specified by the programmer.

Since there may be a wide distribution in the workload, a simple *static priority* method based on an average workload is unsatisfactory. Under heavily loaded conditions, there would be a high likelihood that timing violations would occur. Therefore, a dynamic prioritizing function based on up-to-date load conditions is used. This method ensures that a predictable performance level is maintained while attempting to optimize throughput. To illustrate the use of dynamic priorities, we will present HALOS' dynamic prioritizing function.

3.5.2 Scheduling

There have been many different scheduling algorithms proposed for soft real-time systems. The *earliest deadline scheduling algorithm* is a proven optimal dynamic priority scheduling algorithm [27]. In this algorithm, processing requests are scheduled in the order of their run-time duration with the constraint that all deadlines must be satisfied. This results in a provably optimal *turnaround time* (*i.e.* it produces a minimum average response time) [13].

In spite of its optimality, the earliest deadline algorithm is not used in practice because a direct application of it leads to practical problems that are not readily overcome [27]. One of the practical problems is knowing the stochastic run-times of processing requests in advance. Also, the algorithm assumes that all processing requests are simultaneously available [13]. Neither of these problems can be overcome in the workstation architecture, therefore we must seek an alternative solution. The prioritizing function used in HALOS is a non-optimal but simple method, proposed by Tanenbaum, called the ad hoc method [13].

In our system, the dynamic priorities are assigned to application requests so that the closer a timing deadline is to being missed, the higher its priority becomes. Whenever there are no imminent timing deadlines, assigned priorities are directly proportional to the static priorities set by the programmer. Therefore, under this condition, the ad hoc method operates as a conventional priority-based schedular.

Dynamic operating system state information is an estimate of the operating system load. With this information, the schedular is able to decide whether it will accept a soft real-time application request. If it appears that the timing deadline of a new application cannot be satisfied, the operating system rejects the request, but may try running it later. This ensures that resources are not wasted on an application that cannot finish on time. Processing bandwidth is thus conserved which improves the performance of the system and also reduces the risk that the timing requirements of applications currently running on the system will be violated.

To receive new application requests, HALOS periodically checks the event queues for new tokens which were placed there by the interpreters. The period is set at one second because this is the timing resolution that we selected for our operating system's soft real-time support. During these checks, the entire group of ready to run application requests are prioritized. The advantage of prioritizing in groups is that the relative importance (within the group) of each request can be compared.

3.5.3 Interface Between HALOS and the KBS

The workstation operating system provides both multitasking and multi-inferencing capabilities. For multitasking, HALOS directly starts the non-KBS applications, likewise, for multi-inferencing of KBS applications, HALOS must interact with the inference engine's facility for external control.

To start a problem, HALOS sends the inference engine the predetermined information (application header) and its calculated dynamic priority value for that KBS application. The inference engine's problem schedular uses the dynamic priority to determine how much relative symbolic processing each KBS application will receive. In reply to the HALOS request, the inference engine's facility for external control returns a unique problem identifier.

This internal identifier permits HALOS to make enquiries about the state of each inference and control them (start, stop, suspend). By manipulating these control primitives, HALOS can ensure that all soft real-time requirements are met for all applications running on the workstation.

3.6 The Centralized Operating System Interface (COSI) Layer

The COSI layer of our operating system was developed to satisfy the workstation's requirement for control. An abstract set of primitives that both the KBS and user shell could use for autonomous and regular control was needed. To meet this requirement, we followed an approach analogous to the idea of *device-independent I/O* (input/output) which is used in standard operating system designs [13]. In particular, we established a set of *sufficiently abstract* primitives so that *complex controlling* commands could be easily specified through software. In addition to this, we provided these primitives with *autonomous exception handling* capabilities to autonomously deal with the occurrence of error conditions. Lastly, we devised a unique method of obtaining service from COSI to satisfy the KBS's need for a centralized access.

3.6.1 Device Independent I/O

Device-independent I/O is a layer of software that controls low-level device drivers and provides higher-level software with a set of abstract I/O primitives [13]. The selection of abstract I/O primitives provided by a device-independent I/O layer is up to the developer's discretion [29]. A criterion normally used is to choose primitives that will be commonly needed. The primitives in such a layer serve two purposes.

The first purpose is to provide device abstraction so that devices are easier to use [29]. For instance, the ability to access data from a hard disk is made easier by the operating system's support of a primitive such as *read.* Rather than have to worry about things like controlling a disk motor, head positioning, and formatting information, the user can rely on the abstraction provided by the *read* primitive.

The second purpose of device-independent I/O software is to provide a common method of access so that all similar I/O operations are specified with the same command

[29]. For example, an operation such as *open* applies to devices ranging from files to data connections. Exploiting this conceptual similarity, a single function call defines each class of I/O operation while additional parameters specify the device. This common access simplifies I/O operations for programmers.

A device-independent I/O layer serves one other purpose, it provides basic error detection and reporting [13]. This service is needed by the operating system so that reliable I/O operations are possible.

With these concepts in mind, we now examine how the COSI architecture builds on the idea of device-independent I/O.

3.6.2 COSI Architecture

COSI provides centralized access to workstation interfaces in a manner conceptually similar to device independent I/O software. The *access point* to COSI is a shared memory region. When control of an interface is required, a request is sent to COSI. The request is a pointer to the data region where additional information for specifying the request is located. The data region contains information such as the type of service needed, relevant parameters, and exception handling specifications.

The need for such a centralized access point stems from the fact that expert systems have difficulty supporting procedural knowledge. As a result, the expert system software relies on the operating system (through COSI) to provide it with procedural processing that is easy to access. In addition, this lack of procedural knowledge also requires that COSI's interface primitives be abstract and that the set of primitives encompass all of the KBS's interface demands.

An intelligent set of workstation interface primitives is provided by COSI. Similarly to the way device-independent I/O software makes access to information on a hard disk easy, COSI extends this use of abstraction to all workstation interfaces.

For example, consider an expert system application which needs to obtain information from an external database, such as the current value of the Canadian dollar at the Montreal Stock Exchange. The request to COSI would identify the operation (database query), the location of the database (MSE), as well as the search topic (value of Canadian dollar). COSI would satisfy this request by issuing a series of specific commands to the communication interface software.

In this example, an ISDN data call would first be established between the workstation and the database location. A command to the ISDN access device would be sent specifying that a D-channel, packet-switched call should be made. Next, the search topic would be transmitted in a packet to the database. Once the requested information was received, the ISDN D-channel connection would be terminated. The new information would then be returned to the application through the shared data region.

An important feature of our workstation is its support for autonomous control. To achieve this, we must exploit the primitive error detection and reporting capabilities of the workstation's device-independent I/O software. Building on these basic error mechanisms, COSI extends the operating system's exception handling capabilities so that unsupervised execution is reliable.

The implementation of autonomous exception handling centers on the ability of devices to detect and report error conditions. Because of this ability, COSI is able to autonomously respond to exception conditions by starting particular routines that take corrective action. These routines are called *autonomous exception handlers*. There are two types of exception handlers: *default* and *custom*.

Default exception handlers, as the name implies, are automatically available any time a request is made to COSI. By providing this, COSI removes the programming burden of explicitly specifying exception handlers for each request. Exception handlers are needed in our system to allow application programmers to write *error-tolerant* autonomous control applications. Without default handlers, programmers would have to write a large number of additional routines to deal with each possible exception condition. For situations where a particular default handler is deemed inappropriate, a programmer may override COSI's exception handler. Thus exception handling in COSI is both powerful and flexible.

Custom handlers can bypass some or all of COSI's default handlers. To accomplish this, the programmer would modify the handlers that are called for particular error conditions that should not use the default handlers. Consequently, when one of these error conditions occurs, the custom handler is started instead of the default. With this facility, the programmer may tailor COSI's exception handling mechanism to meet special system requirements. The importance of error handling is illustrated in the following example.

Consider an autonomous ISDN file transfer application that uses a floppy diskette to store the data. One of the default handlers for storing data on a floppy disk can recover from an overloaded disk condition. If a shortage of disk space occurs during this unsupervised file transfer, then the default handler is activated. The exception handling routine closes the primary data file on the floppy diskette, then opens a secondary file on a hard disk to receive the remainder of the file transfer. Without autonomous exception handling, this application would have halted due to the error condition and left the costly communication channel open. Hence, the COSI architecture is providing crucial autonomous exception handling support so that autonomous control is obtainable.

CHAPTER 4

An Experimental ISDN Personal Workstation

We developed an experimental ISDN Personal Workstation to demonstrate a possible implementation of the architecture proposed in Chapter 3. The experimental setup consisted of both off-the-shelf and custom software and hardware components. The criteria for selecting these components were based on adhering to the requirements and capabilities of our proposed workstation architecture.

In Chapter 3, we showed that the operating system of an autonomous multitasking workstation must be event-driven and multitasking. Furthermore, to implement an event-driven architecture, we needed strong interrupt support. The multitasking criterion eliminated all single tasking operating systems as potential candidates. The operating system we chose was the iRMX II.3 286/386 operating system[1] because it satisfied both criteria.

For purposes of flexibility and future expandability, we wanted a platform that supported the popular Industry Standard Architecture (ISA) expansion slot interface. This interface would allow us to either build or purchase hardware modules for the workstation. Figure 10 illustrates how the workstation could be used to support various applications. In this figure, solid boxes represent developed applications and dashed boxes correspond to possible future applications.

We also required our workstation to support CPU intensive symbolic processing applications and therefore needed a CPU with sufficient processing strength. A platform

[1] iRMX is a copyright of Intel Corporation.

which supported the ISA bus architecture and could run the iRMX operating system is the Intel System 120. In the System 120, processing is done on a 32-bit 80386 microprocessor and an 80387 numerical coprocessor, which meet our CPU requirements.

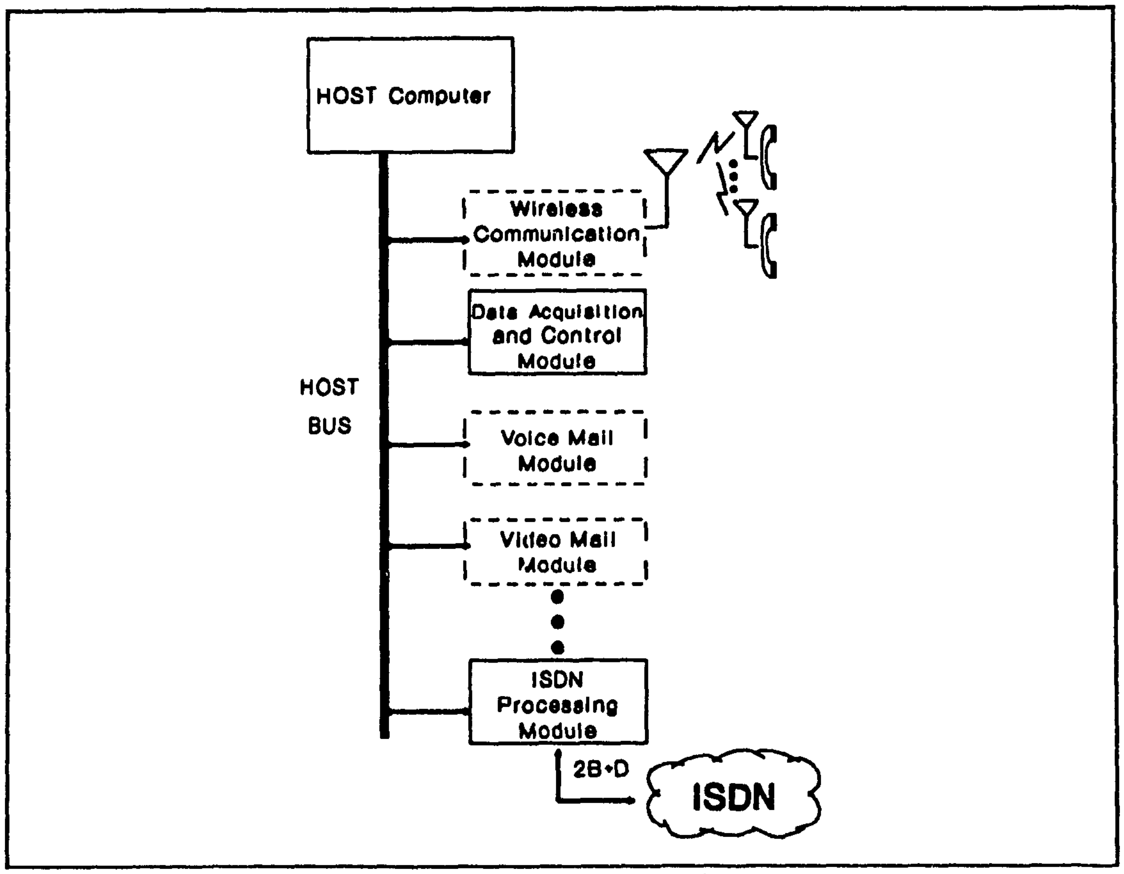


Figure 10 System Level View of the ISDN Workstation.

We required an ISDN Basic Rate Interface (BRI) [12] connection for the workstation. This necessitated hardware and software corresponding to OSI layers 1 to 3. Further, we wanted the processing of layers 2 and 3 to be done on a separate card to preserve CPU bandwidth. The card which met most of the requirements was the ISP188

ISDN Basic Access Product[1]. This card has an on-board 80188 microcontroller and private memory space that is used to run the ISDN software.

The main disadvantage of the ISP188 product was that it required a single tasking operating system environment (*i.e.* DOS) to run on. This left us with the problem of integrating the ISP188 ISDN product with the System 120. We solved this problem by running the ISDN board in DOS on an IBM PC and then linking the two computers with a digital connection.

To implement the digital connection, we developed a Parallel Data Link Control (PDLC) software and hardware product. PDLC provides communication facilities that allow the host computer (System 120) to obtain ISDN services from the ISP188 product in the IBM PC.

4.1 Experimental Hardware Setup

The major block in our experimental setup is the *ISDN Personal Workstation* which comprises a System 120 and an IBM PC. These two platforms are integrated using our PDLC as depicted by the left block in Figure 11. The other block is the *ISDN NT simulator*, which is used to simulate and end-to-end ISDN information exchange for the workstation. The workstation is connected to the NT simulator through an ISDN S-bus [30] interface.

[1] ISP188 ISDN Basic Access Product is a copyright of DGM&S Inc. and Intel Corp.

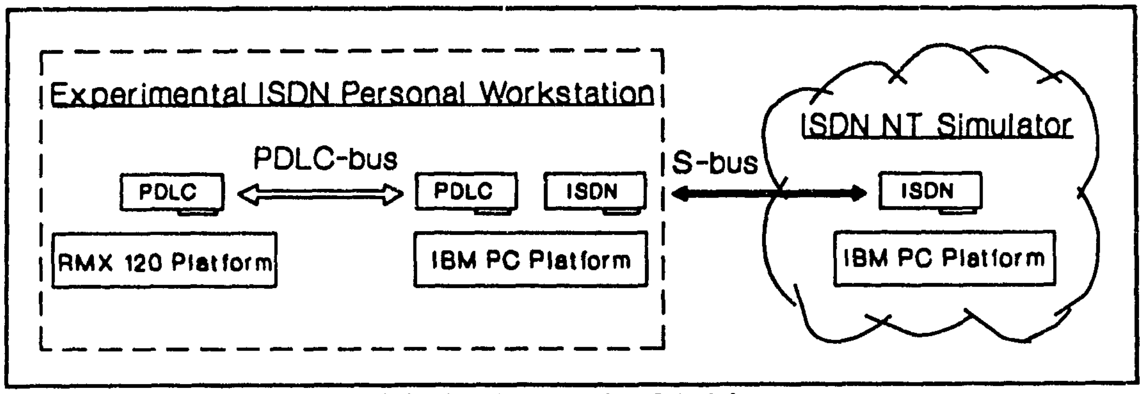


Figure 11 Hardware View of the Implementation Model.

4.1.1 PC Architecture

The IBM PC architecture is an *open architecture* largely because it extends its internal system-bus [31] to a set of expansion slots. These expansion slots allow additional peripheral boards to be added to the system. The electrical signals used by the system-bus are well defined which permits third-party developers to build boards that can easily integrate into the architecture. Following the ISA expansion bus standard, many feature boards have been developed for this architecture.

4.1.2 ISP188 ISDN Basic Access Product Overview

The ISP188 ISDN BRI hardware and software product that we used simulates an end-to-end (TE-to-TE) type of information exchange as though an intervening network were present [32]. The package consists of two ISA expansion cards and accompanying software designed for an IBM PC DOS platform. With these two cards, a connection between a terminal equipment (TE) [30] and a network terminator (NT) [30] can be simulated.

One board implements the TE element of the ISDN reference configuration model at the S reference point [30]. The second board simulates NT functions for only a single ISDN S-bus connection. Both boards interact with the PC host system through an on-board shared memory location. Their DOS device driver controls the interface to these memory locations. During initialization, the boards are downloaded with 80188 binary software that provide up to ISDN layer 3 protocol support for NT or TE functions.

The ISP188 ISDN Basic Access Product also supplies an application library for user application software written in the C language. Some of the functions provided by this application library include: connecting and disconnecting calls; transmitting and receiving data; controlling handset operations; and obtaining ISDN connection status information. From these basic library functions, we developed our own ISDN workstation applications.

4.2 Experimental Software Setup

We implemented the ISDN Personal Workstation as a distributed computer system that uses two operating systems. All application processing is performed in the iRMX operating system on the System 120. ISDN processing, however, is performed in DOS on the IBM PC. In this section we will discuss the iRMX operating system, the layered organization of the workstation operating system, and the implementation of the major layers.

4.2.1 iRMX Operating System Overview

The iRMX II.3 286/386 Operating System that we used in our implementation uses an 80286 or 80386 processor in protected virtual address mode. It is an object-oriented, multitasking operating system with a pre-emptive, priority-based schedular [33]. For equal-priority tasks, it uses round-robin scheduling with time quotas. Intel designates iRMX as a real-time operating system because it supports hardware interrupts which can preempt the kernel within a maximum interrupt latency. In section 2.2 of this thesis, we defined this class of service as *hard real-time*. The value of the maximum interrupt latency was measured to be approximately **76.9** μs. Appendix 1 lists our test measurements.

The iRMX operating system consists of seven layers, each of which provides features that can be used in an application [33]. When establishing a system configuration, all layers are optional except for the nucleus layer. The caveat is that some of the remaining six layers require services from the other layers of the operating system. This sometimes necessitates the inclusion of additional layers in the complete system. We list these dependencies as well as the memory requirements of each layer in Table 1.

We will next briefly describe each system layer. The complete description of iRMX can be found in references [34,35].

(i) Nucleus

The nucleus layer is the core of the operating system and is responsible for task management. Some of the features provided by the nucleus include: multitasking; pre-emptive priority scheduling; round-robin time quota scheduling; memory management;

as well as intertask communication and synchronization using mailboxes and semaphores. Building on these features, the rest of the operating system is established.

Table 1. iRMX Operating System Layer Characteristics [36].

System Layer	Required Layers	Code Size (KB)	Data Size (KB)
1. Nucleus	none	34	2
2. BIOS	1	97	0.108
3. EIOS	1,2	19	0.016
4. UDI	1,2,3,5,7	9	0.032
5. Application Loader	1,2,3	10	0.1
6. System Debugger	1	35	1
7. Human Interface	1,2,3,5	84	0.224

(ii) Basic I/O System (BIOS)

The BIOS layer contains device drivers for reading from and writing to peripherals, as well as the ability to buffer I/O. The BIOS also contains commands for using the iRMX file system and provides access to peripherals through a standard device driver interface.

(iii) Extended I/O System (EIOS)

The EIOS layer provides services similar to the BIOS, with simplified calls that give less explicit control of device behaviour and performance. The EIOS also permits multiple access to a shared device through a logical name. This supports sharing of devices in a multitasking environment.

(iv) Universal Development Interface (UDI)

The UDI layer provides a standard set of system calls which allow applications to run on any operating system supporting the UDI standard. For example, many of the C-286 libraries depend on the UDI layer.

(v) Application Loader

The application loader can load object files into memory from secondary storage under the control of the operating system. It can load non-relocatable code into fixed locations, relocatable code into dynamically allocated memory locations, and load files containing overlays.

(vi) System Debugger

The system debugger is used to debug applications and give a view into the system itself.

(vii) Human Interface (HI)

The HI layer allows multiple users to develop applications, maintain files, run programs, and communicate with the operating system. It consists of a set of system calls, a set of commands, and a command line interpreter. Commands are available for file management, device management, and system status.

4.2.2 Tailoring iRMX

The iRMX operating system can be tailored to meet specific memory and performance requirements. We tested the idea of reducing the size of the operating system by removing non-essential layers. We also tested the idea of making an application program part of the operating system so that it is loaded during boot-up.

To customize the operating system, we used the iRMX Interactive Configuration Utility (ICU) [37]. With this utility, we modified the default *definition file*. This file contains the following information: initialization parameters; selected operating system layers; selected device drivers; and, the name of our application job. The software for the target system is stored in the *load file* so that the bootstrap loader can use it during system initialization.

The function of the bootstrap loader is to load a target system into RAM from secondary storage so that it can begin executing. The iRMX operating system divides the bootstrap loading process into three stages.

The first stage resides in ROM and determines the bootstrap device and the name of the file to load. It contains a set of minimal device drivers used by the first and second stages so that the bootstrap device containing the system can be read. The last function performed by the first stage is to load part of the second stage and transfer control to it.

The second stage is non-configurable, application-independent software. Once it loads itself into RAM, it then loads the third stage and relinquishes control to it.

The third stage loads the application job and then switches the CPU into protected virtual address mode. After this, the application is started.

In Appendix 2, we provide the program listing of our test application job. The test application is classified as an *I/O First Level Job* because it is loaded during initialization and has access to system calls above the BIOS layer. The starting address of the application is specified under the task entry point parameter of the ICU. This information is used by the nucleus to obtain the selector and offset of the initial task, as well as its data selector, and the starting address of an optional custom exception handler.

Through this experiment, we demonstrated how to customize the operating system. Our tailored operating system consisted of the nucleus, BIOS, EIOS, and debugger layer, as well as the application job. The achieved reduction in size was from 330 KB (when all layers were included) to 220 KB. The remainder of our implementation work used the full iRMX operating system because it facilitated development. However, any final application system could clearly be made into a customized operating system.

4.2.3 Software Organization

The layer organization of the experimental workstation's operating system is shown in Figure 12. This drawing illustrates the integration of the iRMX operating system with the theoretical event-driven architecture presented in Chapter 3[1]. The shaded regions represent iRMX operating system layers. We used iRMX to implement most of the lower layers of the theoretical architecture and custom implemented the top three layers.

[1] Note that only the most important layers are shown and that we excluded the Application Loader, HI, UDI, and System Debugger layer from the diagram.

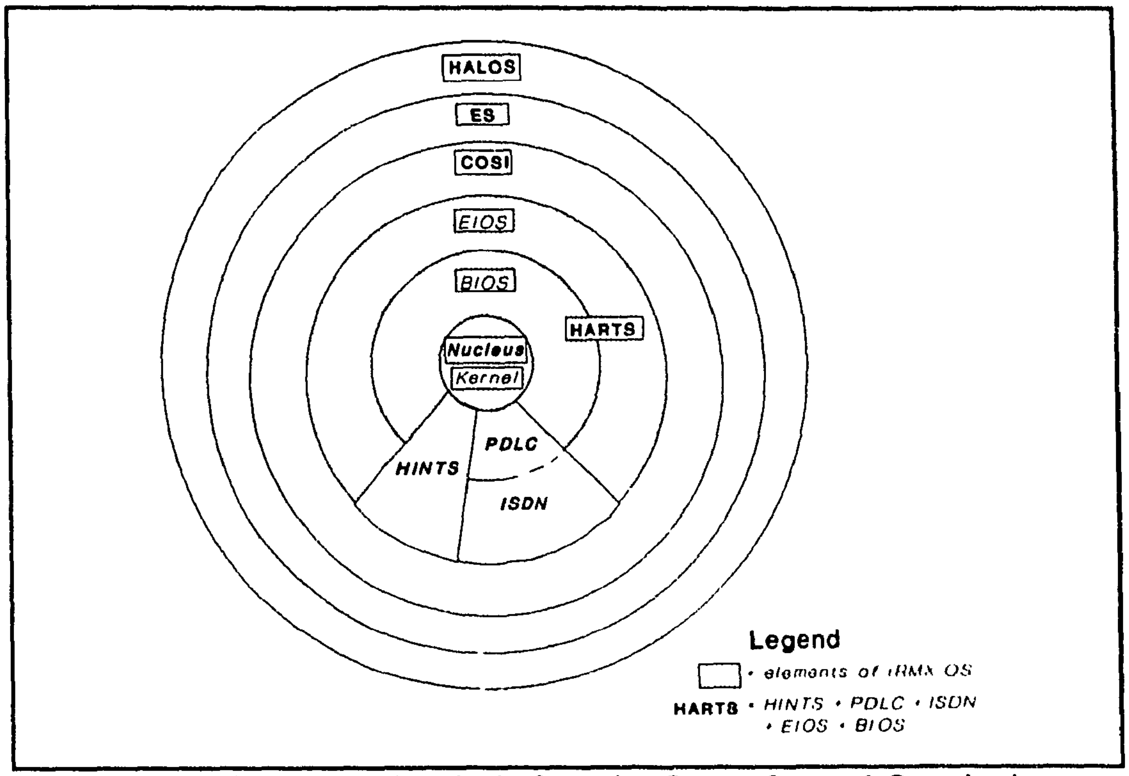


Figure 12 Experimental Workstation's Operating System Layered Organization.

The theoretical HARTS layer comprises the two iRMX I/O layers and two custom modules. The first module is called Hardware Interrupt Support (HINTS). The HINTS software is used to support the hardware interrupt generator that we built to simulate OS events. The schematic diagrams for this hardware is provided in Appendix 3. A detailed description of the second module (PDLC/ISDN) will be made in sections 4.4 and 4.5.

Before discussing the top three layers that we implemented, let us examine their synchronization diagram shown in Figure 13. In this diagram, all the tasks which make up our implementation of the experimental operating system architecture are shown along with the message passing system. All mailboxes, semaphores and task switching facilities are provided by iRMX.

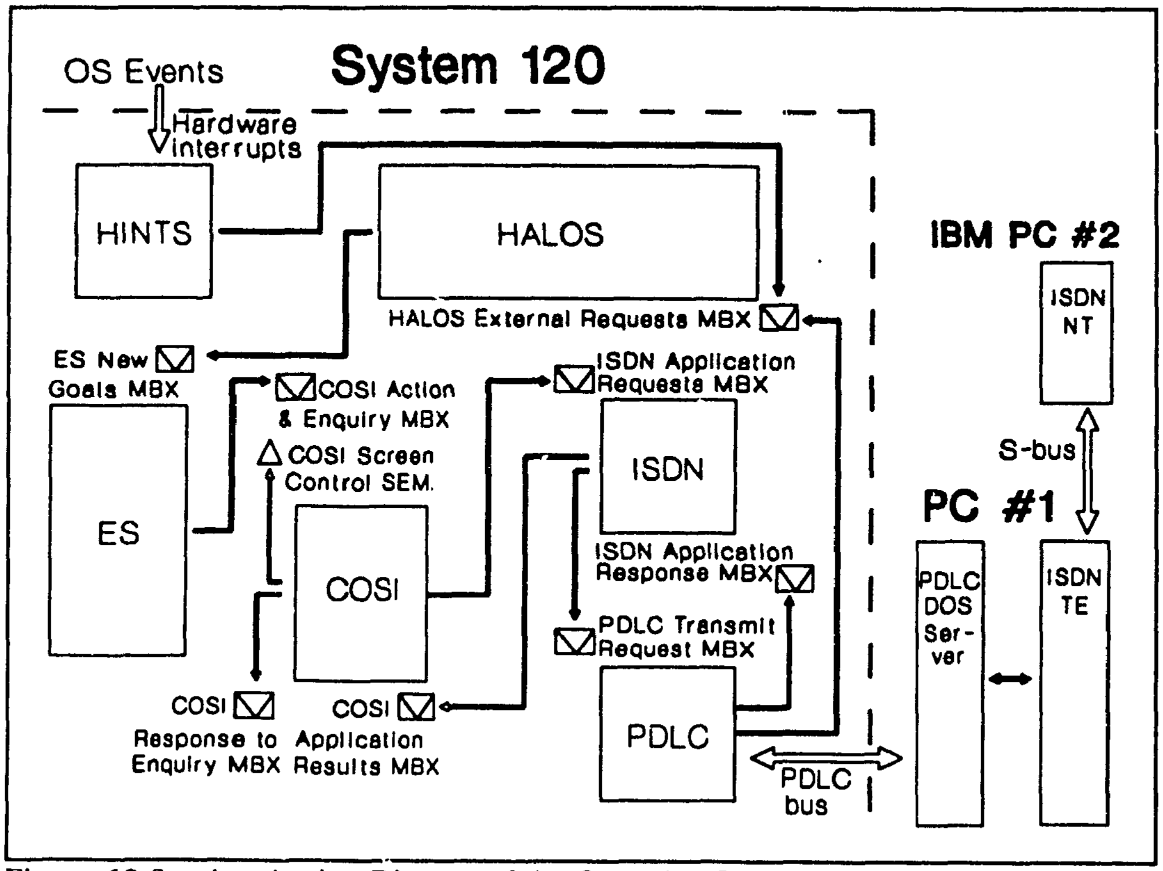

System 120
OS Events
Hardware interrupts
HINTS
HALOS
IBM PC #2
HALOS External Requests MBX
ES New Goals MBX
COSI Action & Enquiry MBX
ISDN Application Requests MBX
ISDN NT
COSI Screen Control SEM.
ISDN
S-bus
ES
PC #1
COSI
ISDN Application Response MBX
PDLC DOS Server
ISDN TE
PDLC Transmit Request MBX
COSI Response to Enquiry MBX
COSI Application Results MBX
PDLC
PDLC bus

Figure 13 Synchronization Diagram of the Operating System.

In the synchronization diagram, the software is organized into six functional blocks: **HINTS**, **HALOS**, **ES**, **COSI**, **ISDN**, and **PDLC**. These blocks implement the major concepts developed in Chapter 3. However, all the features that were defined are not implemented in this scaled-down experimental setup. The pseudo-code for the experimental implementation of the COSI, ES and HALOS blocks of the operating system will be discussed next.

COSI provides the knowledge-based system with intelligent workstation interface primitives. The ES obtains ISDN services by sending a token specifying the desired service to COSI. COSI processes the request by sending other token(s) to the ISDN software. The pseudo-code for COSI is shown below:

```
Loop forever

    1.    Wait at the WCOSI$COMMAND$DMB mailbox for a token specifying
          the desired service.

    2.    Based on the value of the token (desired service), branch to the correspon-
          ding routine.

    3.    Perform the requested service. Send a semaphore unit to WCOSI$READY-
          $1US when control of the screen can be returned to the ES.

          3.a.   If the requested service is an ENQUIRY then place the reply in
                 WCOSI$RESPONSE$DMB mailbox.

          3.b.   If the requested service is an ISDN ACTION then send appropriate
                 command(s) to the ISDN software through the WISDN$COM-
                 MAND$DMB mailbox. If necessary, wait for a reply to the
                 command at the WISDN$STATUS$DMB mailbox.
End loop.
```

The pseudo-code for the ES running in autonomous mode is given below. The pseudo-code describes the operating system's point of view of the ES task. For a detailed explanation of the symbolic processing the ES performs, we refer you to reference [16].

```
    1.    Wait at the WES$COMMAND$DMB mailbox for a goal identifier.

    2.    Repeat until the problem identified by the goal is solved.

          Do one of the following:

          2.a.   Process rules.

          2.b.   Request information from the operating system by sending an
                 ENQUIRY token to COSI in the WCOSI$COMMAND$DMB
                 mailbox. Wait for a response to the ENQUIRY at the WCOSI-
                 $RESPONSE$DMB mailbox.

          2.c.   Request ISDN oriented service from the operating system by
                 sending an ACTION token to COSI in the WCOSI$COMMAND-
                 $DMB mailbox. Wait for a semaphore unit at the WCOSI-
                 $READY$1US semaphore to regain control of the screen.
```

HALOS provides application scheduling for the workstation operating system. In autonomous mode, OS events are identified and then corresponding knowledge-based system applications are started. The pseudo-code for HALOS is shown below:

```
Loop forever
    1.  Wait at the WHALOS$COMMAND$DMB mailbox for an application
        request.
    2.  Based on the application request, start the appropriate knowledge-based
        system application by sending the GOAL identifier to the WES$COM-
        MAND$DMB mailbox in the ES task.
End loop.
```

The remainder of this chapter will discuss the implementation of the PDLC functional block and explain how it was used by the ISDN block. Before we can do this, we must begin with an examination of how the iRMX and DOS interrupt systems work since the manner in which these operating systems handle interrupts greatly influences our implementation.

4.3 Interrupt Systems

Fundamental to efficient processor utilization in a computer system is interrupt processing. An interrupt signal is the mechanism by which an external device can notify the operating system about an event [34]. Unlike in a polling system, to the host computer an interrupt system is completely asynchronous. Interrupts occur only when a peripheral has something to communicate with the host. As a result, computing resources are not wasted on performing synchronous checks. Instead, on demand attention can be obtained from the host processor by generating an interrupt request.

A well supported interrupt system provides applications with a great deal of flexibility to control multiple external activities. In the case of device drivers, all higher level applications can perform computations involving external devices without stopping periodically to poll them. This flexibility facilitates development of large, complicated applications.

Next, we will examine the interrupt systems of the System 120 computer running iRMX and then, the IBM PC running DOS.

4.3.1 iRMX Interrupt System

The iRMX interrupt system consists of both hardware and software components. The hardware components comprises two 8259A Programmable Interrupt Controller (PIC) [38] and an 80386 microprocessor. The software components are the operating system and the user's interrupt programs. We will explain the iRMX interrupt system by following the sequence of events involved when a hardware interrupt occurs[1]. The entire interrupt system is illustrated in Figure 14.

(1) When an external device, such as the hardware component of the PDLC receiver, requires processing attention from the CPU, it raises the voltage level on its physically assigned interrupt request line of the PIC. Each PIC has eight prioritized inputs (levels) which are expanded through cascading. The PDLC's request is latched in the

[1] In addition to the hardware interrupts, there are also software interrupts and trap interrupts, which will not be covered in this discussion. We will use the terms *hardware interrupt* and *interrupt* interchangeably to refer to external hardware interrupts.

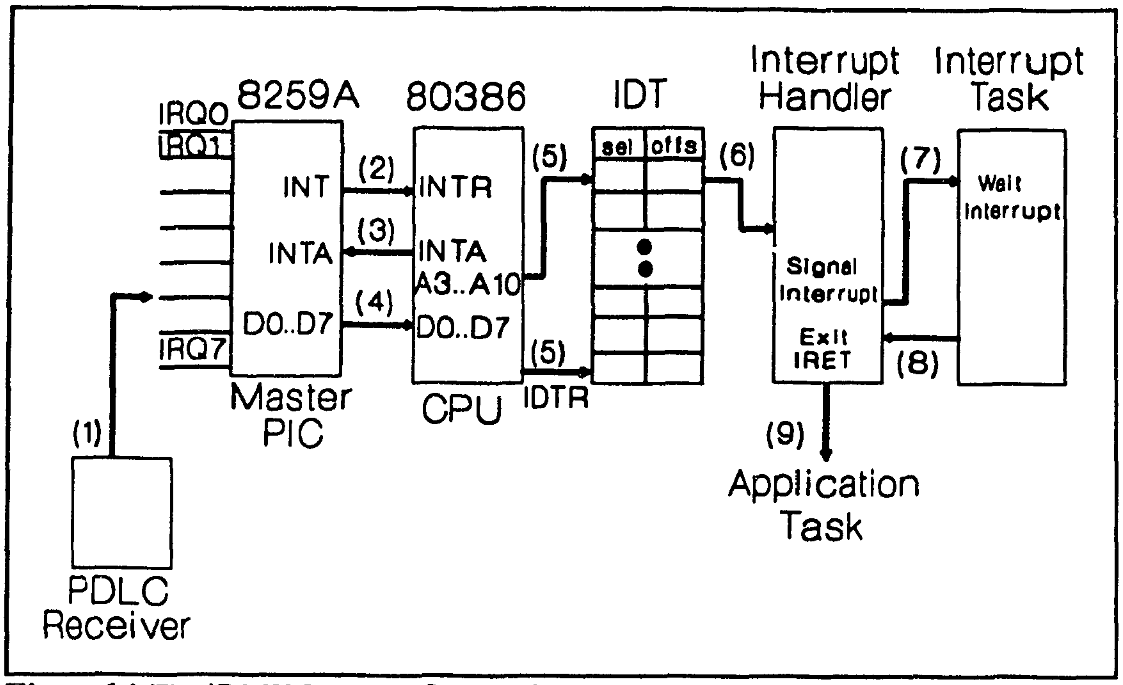


Figure 14 The iRMX Interrupt System for a System 120 Computer.

PIC's interrupt request register (IRR) then fed through priority-resolving logic into the in-service register (ISR). The IRR stores all the interrupt levels that are requesting service (pending) while the ISR stores the interrupt level which is presently being serviced.

(2) The PIC next sends the interrupt request to the CPU using the INT output pin.

(3) The CPU then acknowledges the interrupt request by sending two acknowledge pulses to the PIC using its INTA output pin. The first pulse latches the IRR and causes one of its levels to be strobed into the ISR. The second pulse requests an 8-bit pointer value called an **interrupt vector**.

(4) Upon receiving the CPU's second acknowledgement pulse, the PIC sends the interrupt vector to the CPU. The interrupt vector is the interrupt number identifying which device requested service.

(5) The CPU then accesses a section of memory, known as the Interrupt Descriptor Table (IDT). It uses the interrupt vector to calculate the index (address offset) into the table. On the System 120, the IDT is stored in base RAM and contains pointers to routines called Interrupt Handlers. Pointers are entered into the IDT either when configuring the operating system device drivers or dynamically by executing the Set$Interrupt [35] system command.

(6) After the interrupt handler's pointer (selector and offset) has been obtained from the IDT, the CPU begins executing it. The CPU first pushes the instruction pointer (IP), code segment (CS) and flag register onto the stack before starting the interrupt handler routine. While the interrupt handler is running, the 80386 automatically disables all other hardware interrupts. For this reason, the interrupt handler should not execute for longer than 54.93 ms [39] otherwise the time interval of the system timer interrupt will be exceeded.

The interrupt handler is limited to the types of iRMX system calls it can issue. Only five interrupt-related system calls are permitted. Therefore, if either the processing time of the interrupt request is not small or if non-interrupt related iRMX system calls are required, then the interrupt handler must invoke another routine called an Interrupt Task with the Signal$Interrupt system call.

(7) The interrupt task waits on its interrupt semaphore queue until it is signalled by its corresponding interrupt handler. The Signal$Interrupt [35] call re-enables higher priority interrupts. Its own interrupt is also enabled as long as the number of interrupt requests has not reached its queue size limit. Since higher priority interrupts are enabled,

an interrupt task can be made to run as long as needed, but keeping in mind that lower priority interrupts are still disabled. The interrupt task is also free to use any iRMX system call that it needs. The interrupt task's processing is complete when it again returns to wait on its interrupt semaphore queue.

(8) When the interrupt task blocks on this interrupt semaphore queue, control returns to a ready application with the highest priority. In our example, this application is the interrupt handler which completes its final processing by issuing an Exit$Interrupt and an interrupt return (IRET) command.

When the operating system receives the Exit$Interrupt [35] command, one of two actions takes place. If the operating system needs to re-enable the interrupt level at this time, it sends a non-specific end of interrupt instruction to the PIC. This informs the PIC that processing for the last interrupt is complete. The other possible action occurs if the operating system needs to re-enable lower priority interrupts. RMX would then send a command to the PIC to unmask all of the interrupt levels.

(9) The last instruction executed by an interrupt handler is the IRET command. When the IRET command is issued, the pre-empted application's IP, CS and flag register are removed from the stack. By restoring the old flag register, all interrupt requests from the PIC to the CPU are unmasked because this necessarily is the state prior to starting the interrupt handler.

4.3.2 DOS Interrupt System

The DOS interrupt system also consists of both hardware and software components. The hardware components are the 8259A PIC[1] and the 80x86 microprocessor. The software components are the operating system and the user's interrupt program. We will explain the DOS interrupt system by following the sequence of events involved when a hardware interrupt [31] occurs. The difference between the two operating systems stems from the fact that DOS operates its processor in real address mode while iRMX operates 80286 and 80386 processors in protected virtual address mode. The entire interrupt system is illustrated in Figure 15.

Steps (1) through (4) are identical to the ones discussed in the iRMX Interrupt System section.

(5) The CPU uses the interrupt vector as an index into a table in low memory. This table contains IPs and offset values of Interrup. Service routines. The particular pointer associated with the current interrupt being serviced is obtained from the table. New pointers are entered into this table using DOS interrupt 21h function 25h (Set Interrupt Vector) [40].

(6) Next, the CPU pushes the IP, CS and flag register onto the stack then begins executing the interrupt service routine. When this routine starts running, all interrupts in the system are disabled. Control over enabling and disabling interrupts is available through two microprocessor instructions. All interrupts are enabled using the STI

[1] In an IBM XT type of computer, only one PIC is present. However, in an IBM AT class of computer, there are two PICs present.

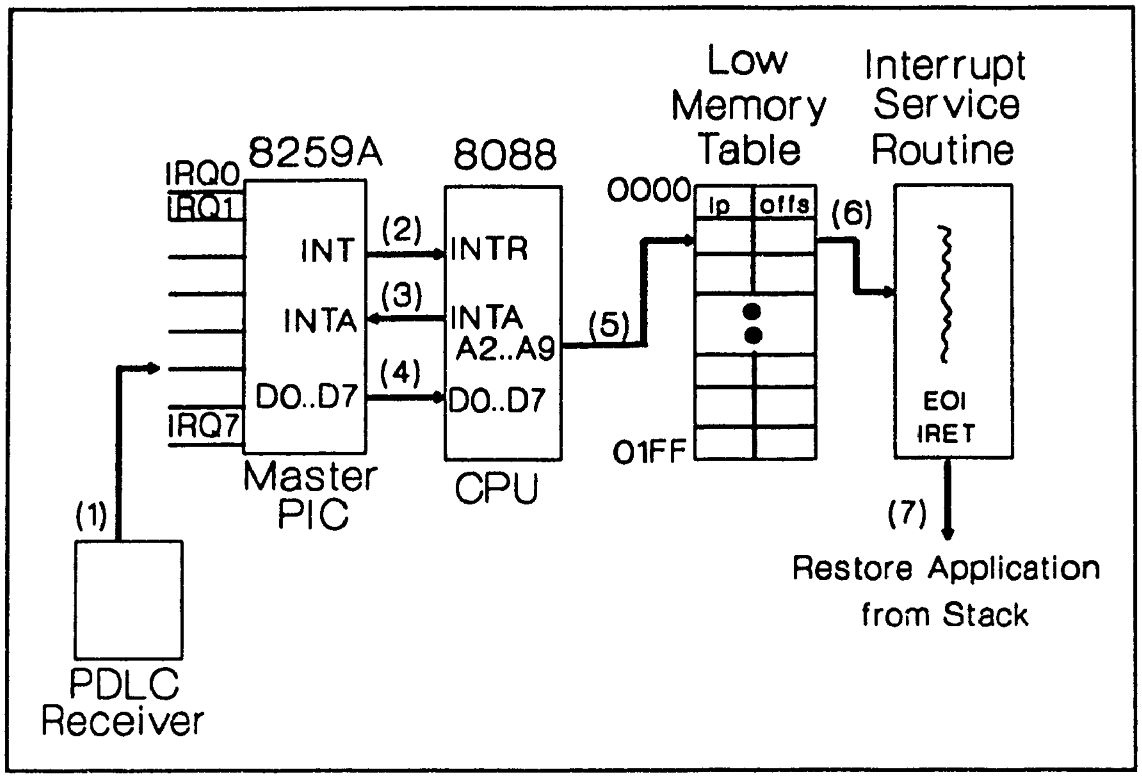


Figure 15 The DOS Interrupt System for an IBM XT Computer.

instruction and disabled using the CLI instruction. It is important to minimize the duration that all interrupts are disabled otherwise certain vital system interrupts will be missed. Before completing its processing, the interrupt service routine sends an end of interrupt instruction to the PIC then does an IRET command.

(7) The IRET command restores the pre-empted application's IP, CS and flag register from the stack. The previously pre-empted application then continues executing once again.

From an application point of view, the main difference between the iRMX and DOS operating systems is the degree of support provided for interrupt processing. In DOS, almost every step is performed without informing the operating system. In iRMX however, the operating system provides much support to facilitate interrupt processing.

Our implementation uses both interrupt systems.

4.4 Parallel Data Link Control (PDLC) Overview

The Parallel Data Link Control (PDLC) is our custom, bi-directional, high-speed, parallel data bus hardware and software product. It is used to provide a layer 2 (**data link layer**) communication link between the System 120 and the IBM PC ISDN server.

The hardware component of our point-to-point custom digital communication system provides layer 1 (**physical layer**) services. Error-free data communication as well as byte-by-byte acknowledgement assures that data is never corrupted or overwritten. Consequently, PDLC layer 2 software can expect the layer 1 hardware to provide it not only error-free communication, but also guarantee that data will never be lost or out of order.

The type of data link layer services provided by the PDLC can be classified as **unacknowledged connectionless service** [17]. This class of service, commonly used in LANs, is appropriate when the error rate is very low and high-speed communication is required [17].

Our layer 2 protocol uses the byte count method [33] to demarcate each frame. We define two types of possible frames: header frames and buffer frames. Header frames range from one to three bytes long and are the default traffic on the bus unless otherwise specified. Header frames carry commands, status information, and parameters such as the size of buffer frames. A buffer frame must follow an appropriate header frame but is not part of the same frame structure. The byte count method is very simple and requires a minimal amount of overhead. It is adequate for our needs and reliable because of the underlying capabilities in our layer 1 hardware.

Flow control is achieved on the PDLC through a combined hardware and software technique that relies on a **backpressure** [33] effect to throttle the flow of data. The hardware buffer, which is directly connected to the parallel data bus, can hold only one byte at a time. The hardware buffer will not issue a clear to send handshaking signal to the transmitting source until its new byte gets removed. Since the transmitter must wait for this signal before strobing the next byte of data, an overrun error will never occur in the hardware. Because the data link layer controls the rate data is removed from the buffer, flow control is attained by applying backpressure on the transmitter *via* the hardware. As a result, PDLC completely controls the rate of data flow.

In addition to layer 2 services, PDLC also provides a frame routing service for incoming messages. This routing service is necessary because multiple logical access to the PDLC is possible. We have defined the following three command/address tokens:

1) non-file ISDN function status

2) ISDN file transfer reception

3) externally initiated ISDN functions.

Using these tokens, PDLC software notifies the appropriate task by mailing the header frame to it. As a result of this routing service, more than one logical PDLC user is possible. This idea is similar to the multiple logical connection services defined for the OSI Transport layer [17]. An abstract view of the PDLC layer that illustrates its separation into router software, data link software, and physical layer hardware; is shown in Figure 16.

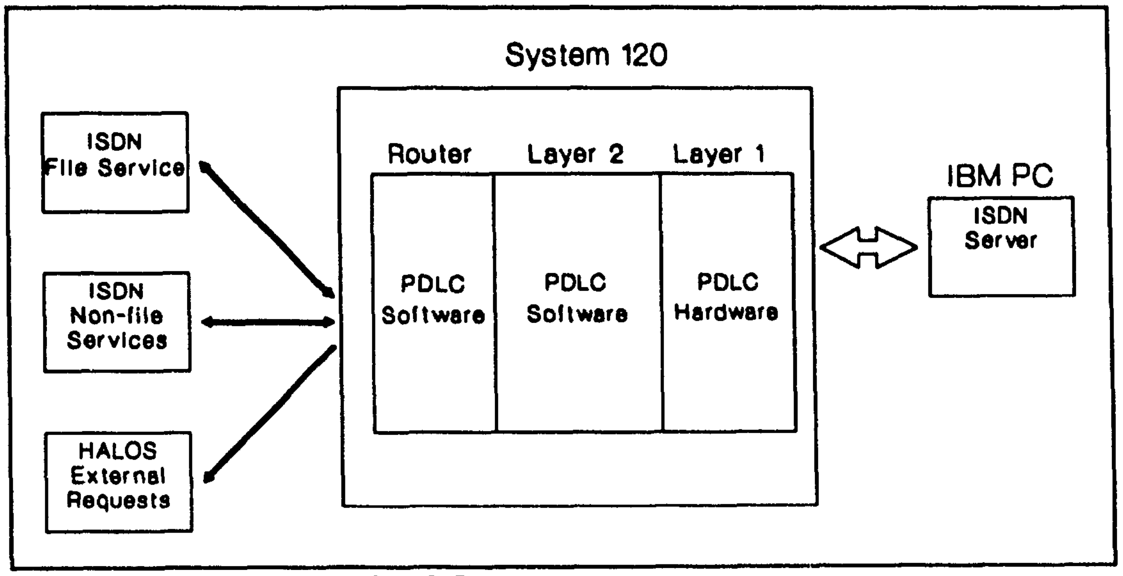


Figure 16 Abstract View of PDLC.

4.4.1 PDLC Hardware Description

The physical layer support for our PDLC software is provided by our PDLC hardware board which is designed for an industry standard XT/AT bus. It is mapped into I/O locations 03A0 to 03AFh. The design is based on the capabilities of a chip called the 8255A Programmable Peripheral Interface (PPI) [34].

We programmed the PPI to operate in mode 1 (strobed input/output). In this mode, fully interlocked handshaking signals are provided for each 8-bit input and output port. The handshaking signals support byte-by-byte acknowledgements which ensure that data is never overwritten. Balance resistors are used to terminate every bus line to improve its transition time. The complete PDLC hardware schematics showing how these handshaking signals are connected are located in Appendix 3.

The bus connection interface was designed so that the PDLC boards in the System 120 and the IBM PC ISDN server would be identical. On the System 120, the receiver and transmitter ports are interfaced to the main processor using hardware interrupt lines. On the IBM PC, the receiver is interrupt driven while the transmitting software uses a polling technique. The complete PDLC hardware system provides high-speed, error-free, digital communication between the two computers.

4.4.2 PDLC Software Description

The PDLC software is interrupt driven in order to give maximum utilization and flexibility to the system. Central to the design are the data structures used for the transmitter and the receiver. The PDLC software elements are shown in Figure 17.

Common to both the transmitter and receiver are the ring buffers which store data. Each ring buffer has two index pointers, one for entering data into the buffer and one for removing data from the buffer. By having two independent pointers, the ring buffer can be simultaneously accessed (*i.e.* data added and removed). The ring buffers are implemented as linear arrays with wrap around indices.

Associated with each ring buffer is an End of Buffer (**EOB**) index pointer. The EOB index is used by the transmitter/receiver interrupt handler to determine when all the necessary data has been sent/received. With it, the interrupt handlers can quickly check the circular buffer without worrying about counters and pointer wrap-around. This technique of using an EOB index simplifies and minimizes the processing performed in each interrupt handler.

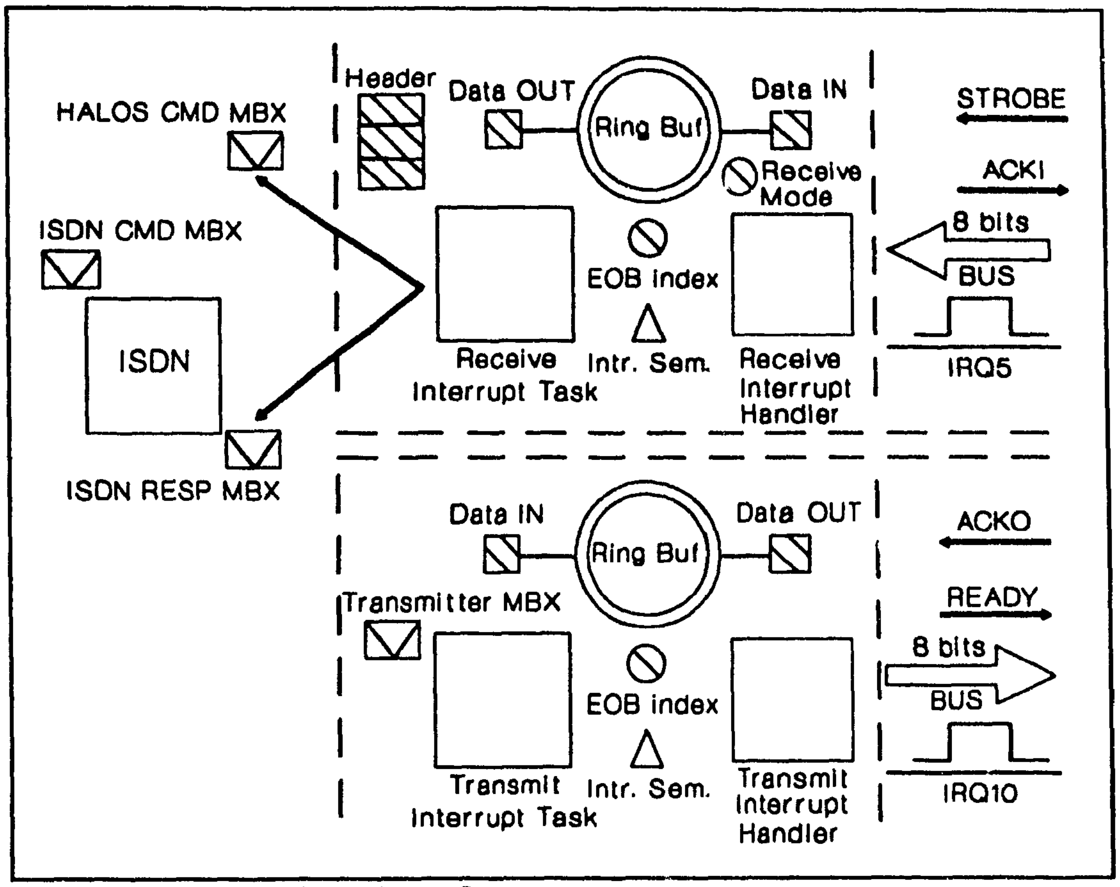


Figure 17 Software View of PDLC.

In addition to the ring buffer, the PDLC receiver also has a three byte array called a **header**. This data structure must be used to receive new commands rather than the ring buffer itself because we wanted the ability to receive new commands without waiting for the previous data to be processed and cleared by a high-level task. Also, since the command information needs to be mailed among tasks, this data structure minimizes unnecessary data movement.

To know when to use the header array rather than the ring buffer, the interrupt handler uses a global variable indicating the receiver mode. We have defined three modes that the receiver can be in: STATREQ, BUFREQ, and EXT$REQ. The first byte of every header frame defines the next mode of the receiver and is stored in the global variable. This variable defines a command/address value that is used to route received frames.

After an ISDN command has been sent, the receiver will be in STAT$REQ mode. In this mode, the receiver may receive a frame which will change its mode to BUF$REQ or EXT$REQ. If however, it is a standard reply (STAT$REQ), the mode will not change. In STAT$REQ mode, status values from previous commands are received in header frames and are then mailed to the ISDN task.

During a file transfer, the receiver is in BUF$REQ mode. The header frame contains the mode identifier and upcoming buffer size. The buffer size is used to calculate the EOB index. The receiver interrupt handler receives data until the EOB value is reached. Next, the receiver interrupt task mails the header information to the ISDN task. This technique allows the ISDN task to independently remove the data from the ring buffer while more new data arrives. In the case of a multiple buffer file transfer, this entire sequence is repeated for each buffer.

The last type of frame is used to notify the operating system (HALOS) of externally initiated ISDN functions. During this type of transfer, the receiver would be in EXT$REQ mode. For example, an incoming ISDN call would send a header frame containing the mode as well as the ISDN service required. The header frame would then be mailed to HALOS for scheduling.

When a task wishes to transmit data using the PDLC, it must first load all the data into the transmitter ring buffer. Once this is done, it mails the size of the loaded data to the PDLC transmitter task. This task then calculates the transmitter EOB index and enables the interrupt handler to begin sending.

Our design of the PDLC transmitter permits another task to load its data into the ring buffer while a transmission is in progress (recall that the ring buffer has two pointers). Thus the next data transmission will be ready to start as soon as the interrupt handler completes the current one and receives the updated EOB value.

For a more detailed description of the software, the reader is directed to the token definitions in Appendix 4 and the source code listing in Appendix 5. Next, we will illustrate the activity sequence of the relevant tasks during a file reception and transmission example.

4.4.3 Illustration of PDLC Operation

In the following diagrams, we show the sequence of activity which occurs during a file reception in Figure 18, and transmission in Figure 19.

The figures are drawn as timing diagrams where a high level represents a running task and a low level represents a blocked task. The up arrows on the dashed line represent data entered into a ring buffer while the down arrows represent data removed from a ring buffer.

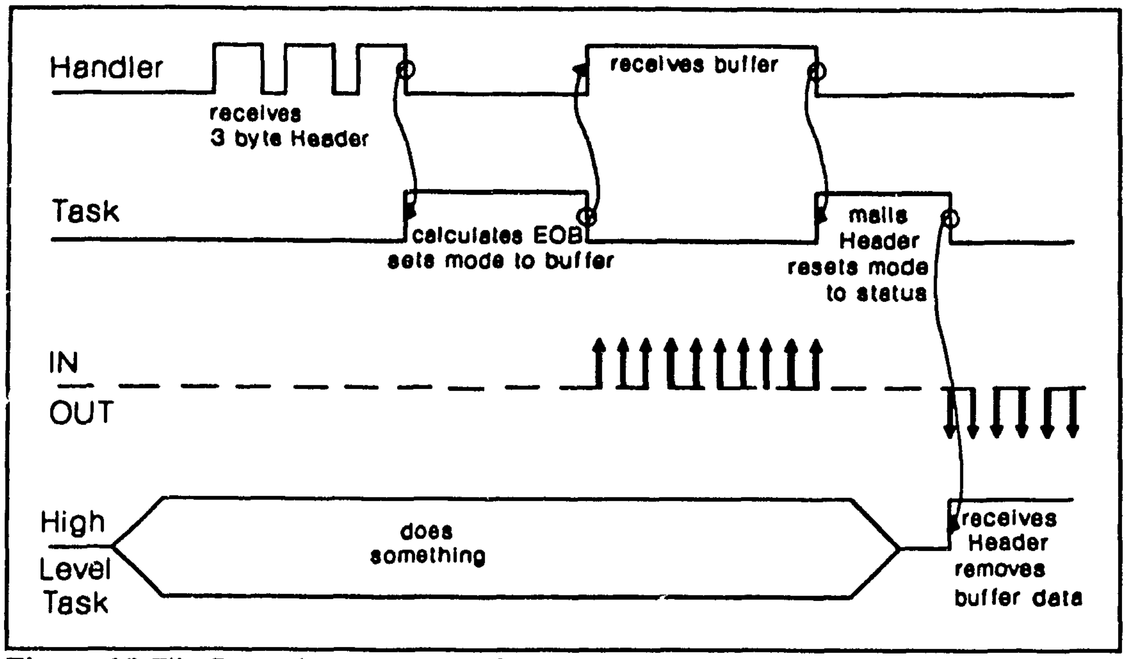

Handler
receives 3 byte Header
receives buffer
Task
calculates EOB sets mode to buffer
mails Header resets mode to status
IN
OUT
High Level Task
does something
receives Header removes buffer data

Figure 18 File Reception using PDLC.

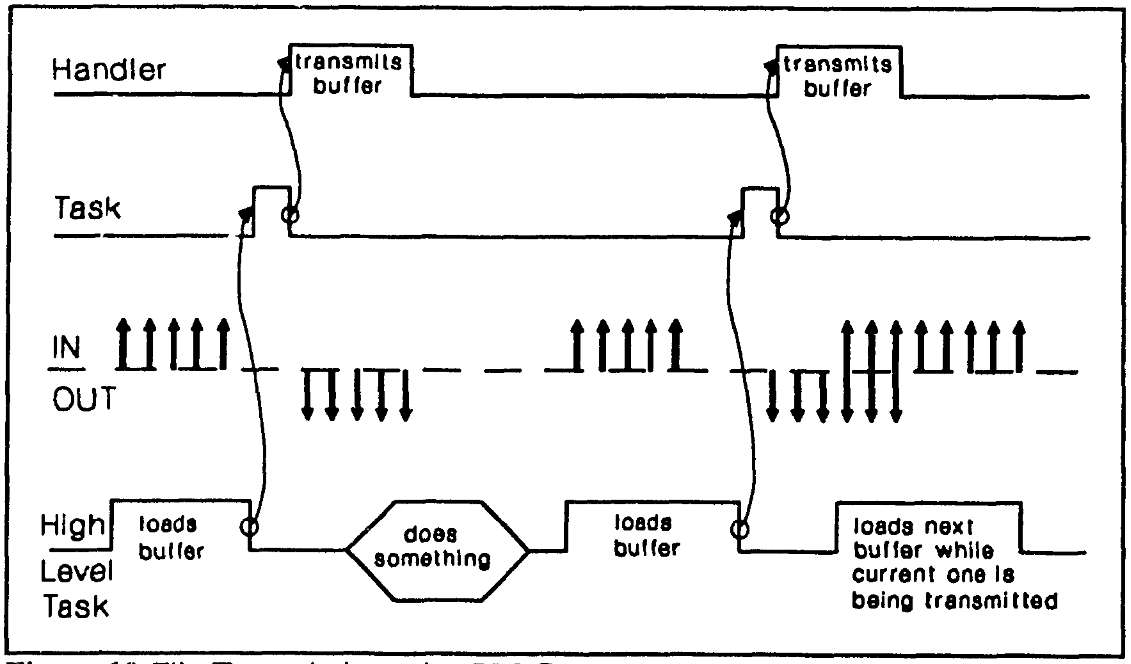


Figure 19 File Transmission using PDLC.

4.5 ISDN Implementation

The ISDN application software running on the System 120 communicates with the DGM&S TE binary software on the IBM PC through a peer-to-peer connection using the PDLC. This arrangement is shown in Figure 20. The System 120 and the IBM PC ISDN Server both contain PDLC hardware and software that supports communication between the higher layer ISDN programs.

Full control of all ISDN applications is maintained by the System 120 but actual ISDN access take place on the IBM PC ISDN server. Using the PDLC, commands and data are sent to the server. The server returns status values and data from the ISDN to the System 120 in the same manner. This distributed computing is not perceived by the end personal workstation user.

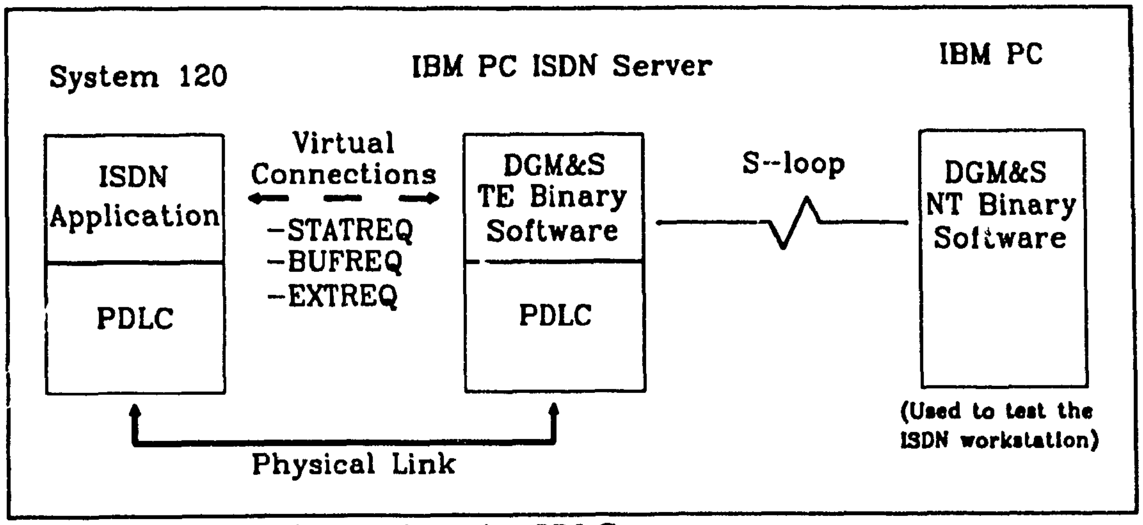


Figure 20 ISDN Implementation using PDLC.

This ISDN application illustrates a possible use of our PDLC. It shows how PDLC may be used to manage the communications required in distributed computing. Any application with a similar digital communication need could exploit this protocol.

CHAPTER 5

Conclusions

5.1 Summary

In Chapter 1, we defined the two motivating features behind the design of the **ISDN Personal Workstation**. We felt users would expect these features from the workstation:

(i) the ability to exploit ISDN information services

(ii) the ability to support autonomous control of the local environment in an intelligent home of the future.

From these features, it was evident that the workstation's operating system architecture would have to support heuristic processing and manage centralized control. In this chapter, we also listed our research contributions.

In Chapter 2, we examined the software architecture requirements needed to support the autonomous control capability. The two requirements were:

(i) **Workstation Interface Primitives**: which are centralized, abstract, and can control the local environment.

(ii) **Scheduling Control**: which supports both hard and soft real-time applications and deals with timing exceptions.

A discussion of both hard and soft real-time systems was also presented.

Chapter 3 described the proposed operating system architecture. The architecture not only supported standard application processing, but was also designed to be event-driven so that the local environment could be autonomously controlled.

The layers of our operating system were explained by first describing how OS events were processed. Next, we examined in detail the functions and design of each operating system layer.

In Chapter 4, we described our experimental ISDN Personal Workstation which demonstrated a possible implementation of the architecture proposed in Chapter 3. We also explained the iRMX and DOS hardware interrupt systems and discussed the implementation software. In particular, we described the design and use of our PDLC.

5.2 Conclusions

The most important conclusions that we can state, based on our research into operating system design and real-time systems, are as follows:

(i) **Autonomous control requires an event-driven operating system design.**

Because of the nature of autonomous control, operations are typically asynchronous. We found that an effective design for dealing with asynchronous operations was the event-driven approach used in network management and control systems. The design of our workstation operating system followed this approach. In our architecture, the HARTS, Interpreter and HALOS layers perform OS event processing. These layers transform OS events into well defined application requests that can then be scheduled by the operating system.

(ii) **Autonomous control requires workstation interface primitives that have autonomous exception handling capabilities.**

In order for unsupervised execution to be feasible, not only must devices detect and report error conditions, but also there must be software that autonomously responds to these exception conditions. We implemented this idea in the COSI layer of our operating system. COSI provides both default and custom autonomous exception handler capabilities. With these, the application programmer can write error-tolerant autonomous control applications on the workstation.

Our experimental ISDN personal workstation demonstrated that the operating system architecture that was presented in Chapter 3 is feasible in the real-world. Our work also showed that most currently available off-the-shelf components are suitable. However, a few problems in integrating the components must be overcome when not using a conventional operating system.

5.3 Suggestions for Future Studies

The following list is a suggestion of implementation work that could be undertaken:

(i) Expand the ISDN application support on the workstation. The next DGM&S ISP188 software release will have an incoming ISDN call capability. Provisions for this capability have already been made in our implementation of the operating system. Packet-switched application support could also be added to the existing ISDN circuit-switched services.

(ii) Design an interface for either the Consumer Electronics Bus Standard or the Smart House Bus Standard so that the workstation could be integrated into an intelligent home environment. These in-home communication standards have been developed so that controllable devices can be networked and controlled by a centralized computer. In this home environment, full use of the workstation's autonomous control features would be possible. The feasibility of such an interface design has already been proven by the development of our PDLC.

(iii) Implement soft real-time scheduling and programming support in the workstation operating system. This feature is essential for autonomous control capabilities and will be needed in future workstation applications.

(iv) Integrate the ISDN board directly into the workstation architecture. To do this, source code must be purchased and device drivers written for the iRMX operating system. These device drivers would then constitute a segment of the HARTS layer.

REFERENCES

[1] N. Shimasaki *et al.*, "An Overview of ISDN - Toward Modern Communications," *NEC Res. & Develop., Special Issue on ISDN*, 1987, pp.3-18.

[2] C.L. Wong and R. Wood, "Implementation of ISDN," *Telesis*, BNR, Vol.13, No.3, 1986.

[3] S.N. Pandhi, "The Universal Data Connection," *IEEE Spectrum*, Vol.24, No.7, July 1987, pp.31-35.

[4] G. Hanover, "Networking the Intelligent Home," *IEEE Spectrum*, Vol.26, No.10, October 1989, pp.48-49.

[5] J. Chatterley, B. Newman and R. Wellard, "The ISDN PC: A Flexible Voice Data Workstation," *IEEE Globecom '86*, 1986, pp.1504-1508.

[6] Y. Langhame and G. Meade, "Smart Buildings: An Emerging Reality?," *IEEE Canadian Review*, No.6, Dec. 1989, pp.6-9.

[7] CCITT, *Integrated Services Digital Network (ISDN)*, VIIIth Plenary Assembly CCITT Recommendations of the Series I Red Book, Vol.3, Fascicle III.5, 1985.

[8] W. Stallings, *ISDN an introduction*, Macmillan, New York, N.Y., 1989.

[9] A.S. Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, N.J., Second Edition, 1988.

[10] B.W. Wah, M.B. Lowrie, and G. Li, "Computers for Symbolic Processing," *Proc. of the IEEE*, Vol.77, No.4, April 1989, pp.509-539.

[11] C.V. Ramamoorthy and B.W. Wah, "Knowledge and Data Engineering," *IEEE Trans. on Knowledge and Data Enginee* No.1, March 1989, pp.9-15.

[12] CCITT, *ISDN User-Network Interfaces, Interface Structures and Access Capabilities*, CCITT I.412 Recommendations, Vol.3, Fascicle III.5, 1985, pp.132-138.

[13] A.S. Tanenbaum, *Operating Systems Design and Implementation*, Prentice-Hall, Englewood Cliffs, N.J., 1987.

[14] T.E. Marques, "A Symptom-Driven Expert System for Isolating and Correcting Network Faults," *IEEE Communications Magazine*, Vol.26, No.3, March 1988, pp.6-13.

[15] M.T. Sutter and P.E. Zeldin, "Designing Expert Systems for Real-Time Diagnosis of Self-Correcting Networks," *IEEE Network*, Vol.2, No.5, Sept. 1988, pp.43-51.

[16] R.D. Rourke, *Programming an ISDN Intelligent Personal Workstation: An Architecture and Language*, M.A.Sc. Thesis, Concordia University, June 1990.

[17] D.A. Waterman, *A Guide to Expert Systems*, Addison-Wesley, reading, Mass., 1986.

[18] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, *Building Expert Systems*, Addison-Wesley, Reading, Mass., 1983.

[19] B. Gates, "The 25th Birthday of BASIC," *BYTE*, McGraw-Hill, New York, N.Y., Vol.14, No.10, October 1989, pp.269-276.

[20] A.K. Mok, *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*, PhD Dissertation, Massachusetts Institute of Technology, May 1983.

[21] D. Vanderlin, "Toward a Real-Time Executive Standard," *UNIX World*, Vol.5, No.4, April 1988, pp.75-81.

[22] E.D. Jensen, C.D. Locke and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems," *IEEE Proc. Real-Time Systems Symposium*, 1985, pp.112-122.

[23] J.A. Stankovic, "Misconceptions About Real-Time Computing," *IEEE Computer*, Vol.21, No.10, October 1988, pp.10-18.

[24] I. Lee and V. Gehlot, "Language Constructs for Distributed Real-Time Programming," *IEEE Proc. Real-Time Systems Symposium*, 1985, pp.57-66.

[25] D.W. Leinbaugh, "Guaranteed Response Times in a Hard-Real-Time Environment," *IEEE Trans. on Software Engineering*, Vol.SE-6, No.1, Jan. 1980, pp.85-89.

[26] C.M. Woodside and D.W. Craig, "Local Non-Preemptive Scheduling Policies for Hard Real-Time Distributed Systems," *IEEE Proc. Real-Time Systems Symposium*, 1987, pp.12-24.

[27] L. Sha, J.P. Lehoczky and R. Rajkumar, "Solutions for Some Practical Problems in Prioritized Preemptive Scheduling," *IEEE Proc. Real-Time Systems Symposium*, 1986, pp.181-191.

[28] I. Lee, R. King and R. Paul, *RK: A Real-Time Kernel for a Distributed System with Predictable Response*, MS-CIS-88-78, Dept. of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Oct. 1988.

[29] D. Comer, *Operating System Design*, Prentice-Hall, Englewood Cliffs, N.J., 1984.

[30] CCITT, *ISDN User-Network Interfaces: Layer 1 Recommendations*, CCITT I.430 Recommendations, Vol.3, Fascicle III.5, 1985, pp.141-177.

[31] L.S. Eggebrecht, *Interfacing to the IBM Personal Computer*, Howard W. Sams & Co., Indianapolis, Ind., 1985.

[32] DGM&S, *ISP188/ISDN Basic Access Product Binary Manual*, Mt. Laurel, N.J., 1989.

[33] Intel, *Extended iRMX II.3 Introduction, Installation and Operating Instructions*, Vol.1, Santa Clara, Calif., 1988.

[34] Intel, *Extended iRMX II.3 User Guides*, Santa Clara, Calif., 1988.

[35] Intel, *Extended iRMX II.3 System Calls*, Vol.3, Santa Clara, Calif., 1988.

[36] Intel, *iRMX II.3 Operating System Product Desription*, Santa Clara, Calif., Feb. 1988.

[37] Intel, *System 120 Software User's Guide*, Santa Clara, Calif., 1988.

[38] Intel, *Microprocessor and Peripheral Handbook*, Santa Clara, Calif., Vol. I, 1989.

[39] IBM, *IBM Technical Reference for Personal Computer AT*, Boca Raton, Florida, 1984.

[40] R. Duncan, *Advanced MS DOS*, Microsoft Press, Redmond, Wash., 1986.

[41] W. Stallings, *Data and Computer Communications*, Macmillan, New York, N.Y., Second Edition, 1988.

[42] Intel, *Microsystem Components Handbook*, Vol. II, Santa Clara, Calif., 1984.

[43] Intel, *OEM Boards and Systems Handbook*, Santa Clara, Calif., 1986.

APPENDIX 1

System 120 Real-Time Performance Measurements

The interrupt latency of the System 120 was experimentally measured as referred to in section 4.2.1. To obtain these results, we used the iRMX operating system in a configuration that included all the system layers. We measured the time interval between the hardware interrupt generator causing a pulse signal, to the interrupt handler executing its first instruction. These experimental values are divided into three groups according to the type of background environment and are given in Table 1.1.

The group 1 consisted of only the interrupt handler and interrupt task running on the operating system. The group 2 consisted of the interrupt software as well as a background environment of (1, 5, 10, 20, 50, 70) tasks having equal priorities. Each task continuously sent then immediately waited to receive a semaphore unit. The iRMX system calls: SEND$UNITS and RECEIVE$UNITS were used. The group 3 measurements consisted of a background environment of 5 tasks which continuously sent then immediately received mailbox data of 128 bytes. The iRMX system calls: SEND$DATA and RECEIVE$DATA were used.

Table 1.1. System 120 Interrupt Latency Measurements.

Interrupt Latency in (μs)								
GROUP:	1	2						3
Test #	0	1	5	10	20	50	70	5
1	15.0	15.3	14.3	14.3	24.4	19.0	27.7	71.6
2	14.3	14.3	15.0	14.3	15.0	14.3	15.0	14.3
3	15.0	13.7	15.0	14.7	15.0	27.0	26.7	15.6
4	14.3	14.7	21.7	16.3	24.4	15.7	26.3	44.9
5	14.7	21.7	17.7	17.0	27.0	14.7	14.3	14.0
6	14.7	16.3	22.3	23.0	25.4	14.3	15.7	66.9
7	14.3	17.3	23.0	15.0	14.3	14.3	15.7	54.3
8	15.0	14.7	16.0	15.0	15.3	14.3	18.0	13.6
9	14.0	14.3	16.3	24.3	15.0	27.0	14.0	16.6
10	14.3	14.3	24.3	15.7	14.3	18.4	14.3	37.0
11	15.0	16.7	23.7	17.7	14.3	15.3	14.3	58.9
12	14.3	17.7	20.3	14.0	13.7	14.3	13.7	19.6
13	14.0	15.7	15.0	14.3	15.0	15.0	14.3	32.6
14	14.3	17.7	14.0	25.0	24.0	17.0	28.4	32.3
15	14.3	14.7	19.7	24.7	15.0	14.3	16.0	15.0
16	15.0	18.7	14.3	16.3	14.3	19.0	16.0	75.6
17	14.3	15.0	29.0	15.0	27.0	15.0	16.3	15.0
18	14.3	27.7	15.7	16.7	23.0	14.3	14.3	42.9
19	15.0	14.0	19.3	13.7	16.7	15.0	25.0	36.3
20	14.3	14.3	14.3	26.0	14.0	17.7	16.3	68.9
21	15.0	15.3	14.3	13.7	13.7	14.7	13.7	76.9
22	14.3	14.3	18.0	20.3	16.3	15.3	22.4	14.3
23	14.0	15.3	15.0	15.7	16.3	15.0	14.3	35.6
24	14.0	14.7	14.3	15.3	15.0	15.0	15.0	56.6
25	14.0	14.3	23.0	15.0	16.0	14.7	15.7	17.6
TYPICAL	14.5	16.1	18.2	17.3	17.8	16.4	17.4	37.9
MINIMUM	14.0	13.7	14.0	13.7	13.7	14.3	13.7	13.6
MAXIMUM	15.0	27.7	29.0	26.0	27.0	27.0	28.4	**76.9**

The experimental results showed that the **maximum interrupt latency** is **76.9** µs. As expected, the results showed that the smallest interrupt latency is obtained when there are no background tasks active. The results also demonstrated that when there are background tasks, then the number of active tasks does not influence the interrupt latency. The interrupt latency is most affected by the particular system call that the background tasks are performing. Hence, for the same system call, an increase in the number of background tasks *does not* produce a corresponding increase in the interrupt latency. The experimental results showed that the mailbox system calls internally disable the interrupts for a longer duration than do the semaphore system calls.

Our experimental results agreed with the specifications provided by Intel in Table 1.2.

Table 1.2. Intel Real-Time Performance Results*.

Interrupt Latency (to handler)	iRMX 286 Execution Time (µs)
Minimum	15.0
Typical	19.0
Maximum	93.0

* The interrupt latency measurement is based upon the interrupt level zero. The background environment included two tasks performing job management primitives and an active interrupt task at level four. The operating system was configured with only the Nucleus layer. [43]

APPENDIX 2

Program listing: I/O First Level Job.

The I/O First Level Job that we wrote to test the idea of tailoring the iRMX operating system is included in this appendix. A discussion of the results is given in section 4.2.2.

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   ISDN Personal Workstation              =
   =    Sub-Project   :   Sample I/O First-Level Job             =
   =    File          :   io1job.c                               =
   =    Author        :   Zenon Slodki, Robert Rourke            =
   =    Start Date    :   05 Apr 1989                            =
   =    Update        :   20 Apr 1990                            =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
/* This program uses only the Nucleus, Bios and Eios layers as well as
   some PLM support routines. No C I/O is used because the UDI layer
   would then have to be included as well.
*/
#include  "/lib/cc286/stdio.h"
#include  "/lib/cc286/udi.h"
#include  "/lib/cc286/rmx.h"

/*
 * Other declaration of C functions to be used
 */
extern  char            *udistr();
extern  unsigned        strlen();
extern  alien unsigned  interr();
extern  alien unsigned  wrterr();

/* PLM routines */
extern      alien unsigned    *Hex2ascii();
extern      alien unsigned    *Hex2dec();

#define LF "\n\r"
#define RMXIOJOB 0

#ifdef      RMXIOJOB
alien       ZENROB()
#else
main()
#endif
{
        char        MessRMX[5];
        char        Mess[25];
        char        MRMX[5];
        char        *Msg0 = "A log of your input is being kept on file.
                    \n\r";
        char        *Msg1          =       "ZenRob Operating System
                    \n\n\r";
        char        *Msg2          =       "C:\\ZenRob>";
        char        *Msg3          =      "Bad command or file name: ";
        char        *Msg5          =       "\n\rYEAR:";
        char        *Msg6          =       "\n\rHOUR:MINUTE:SECOND:";

byte  inarray[80];
token co$comm, co$conn, ci$conn, hd$conn;
word  BYTEW, byte$r, k, status;

/*
 * A structure to recieve the time from the BIOS system
 */
struct      Date_struct
{
      char        seconds;
      char        minutes;
      char        hours;
```

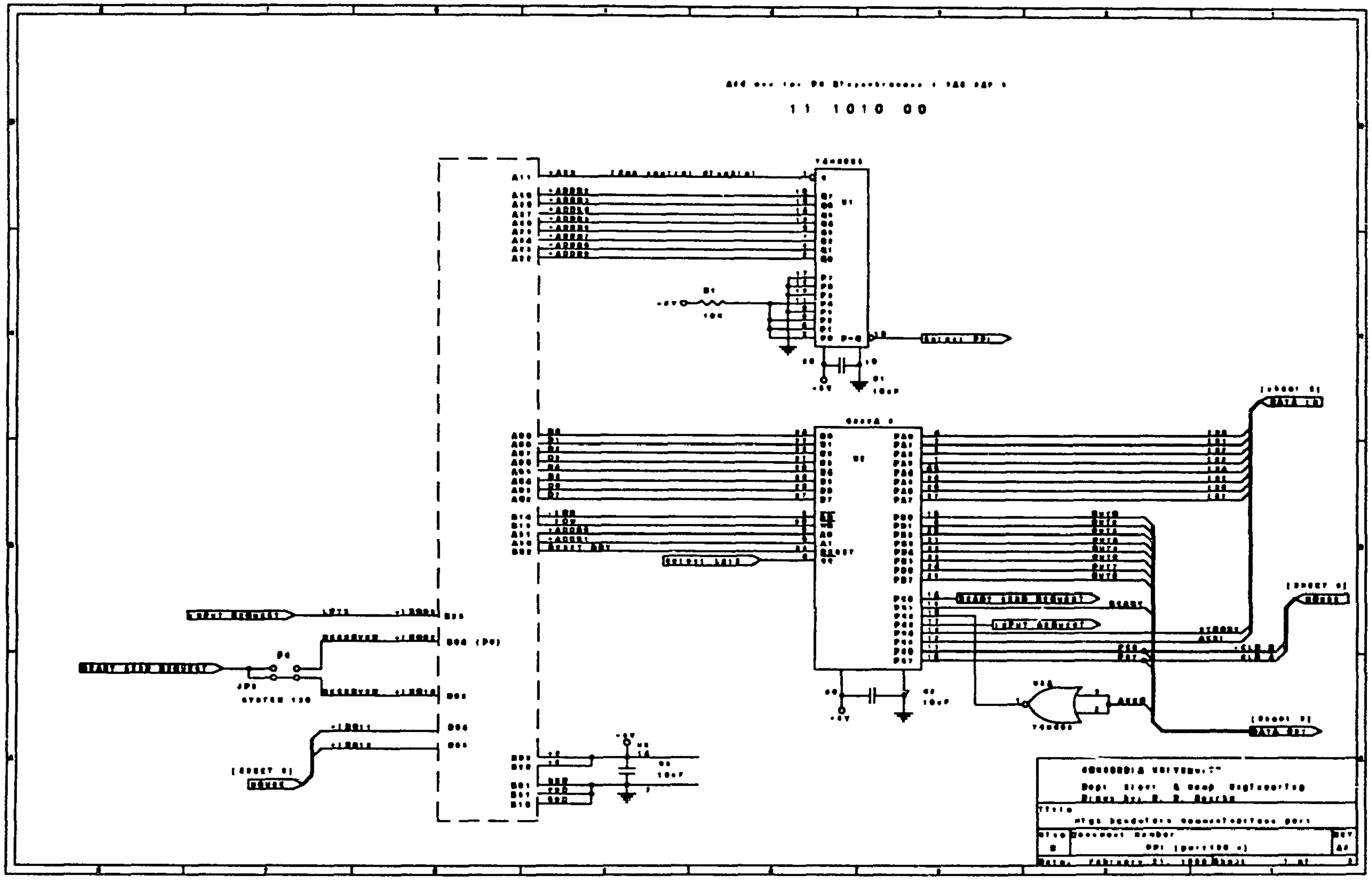
```
        char        days;
        char        months;
        unsigned    years;
};
struct              Date_struct date;
byte  datestr[18];
byte  msg[15];

#define cr  13
#define lf  10
msg[5] = cr;
msg[6] = lf;
msg[7] = lf;
datestr[15] = cr;
datestr[16] = lf;
datestr[17] = lf;

/* associate physical device name to logical connection name */
/* physical device is the default console: terminal & keyboard */
rq$logical$attach$device(udistr(MessRMX,":CO:"),udistr(MRMX,"CON"),1,
        &status);
if (status != 0 )
        interr( status );

/* obtain connection token associated with logical connection */
co$conn = rq$s$attach$file(udistr(MessRMX,":CO:"), &status);
if (status != 0 )
        interr( status );

/* open connection for both input and output, no buffers */
rq$s$open(co$conn, 3, 0, &status);
if (status != 0 )
        interr( status );

/* associate physical device name to logical connection name */
/* physical device is the serial port 1: terminal consol */
rq$logical$attach$device(udistr(MessRMX,":CC:"),udistr(MRMX,"COM1"),1,
        &status);
if (status != 0 )
        interr( status );

/* obtain connection token associated with logical connection */
co$comm = rq$s$attach$file(udistr(MessRMX,":CC:"), &status);
if (status != 0 )
        interr( status );

/* open connection for output, no buffers */
rq$s$open(co$comm, 2, 0, &status);
if (status != 0 )
        interr( status );

ci$conn = co$conn;

/* obtain connection token associated with logical connection */
hd$conn = rq$s$attach$file(udistr(Mess,":SD:user/zen/hdcopy"), &status);
if (status != 0 )
        wrterr( status );

/* open connection for output, 4 buffers */
rq$s$open(hd$conn, 2, 4, &status);
if (status != 0 )
        wrterr( status );
```

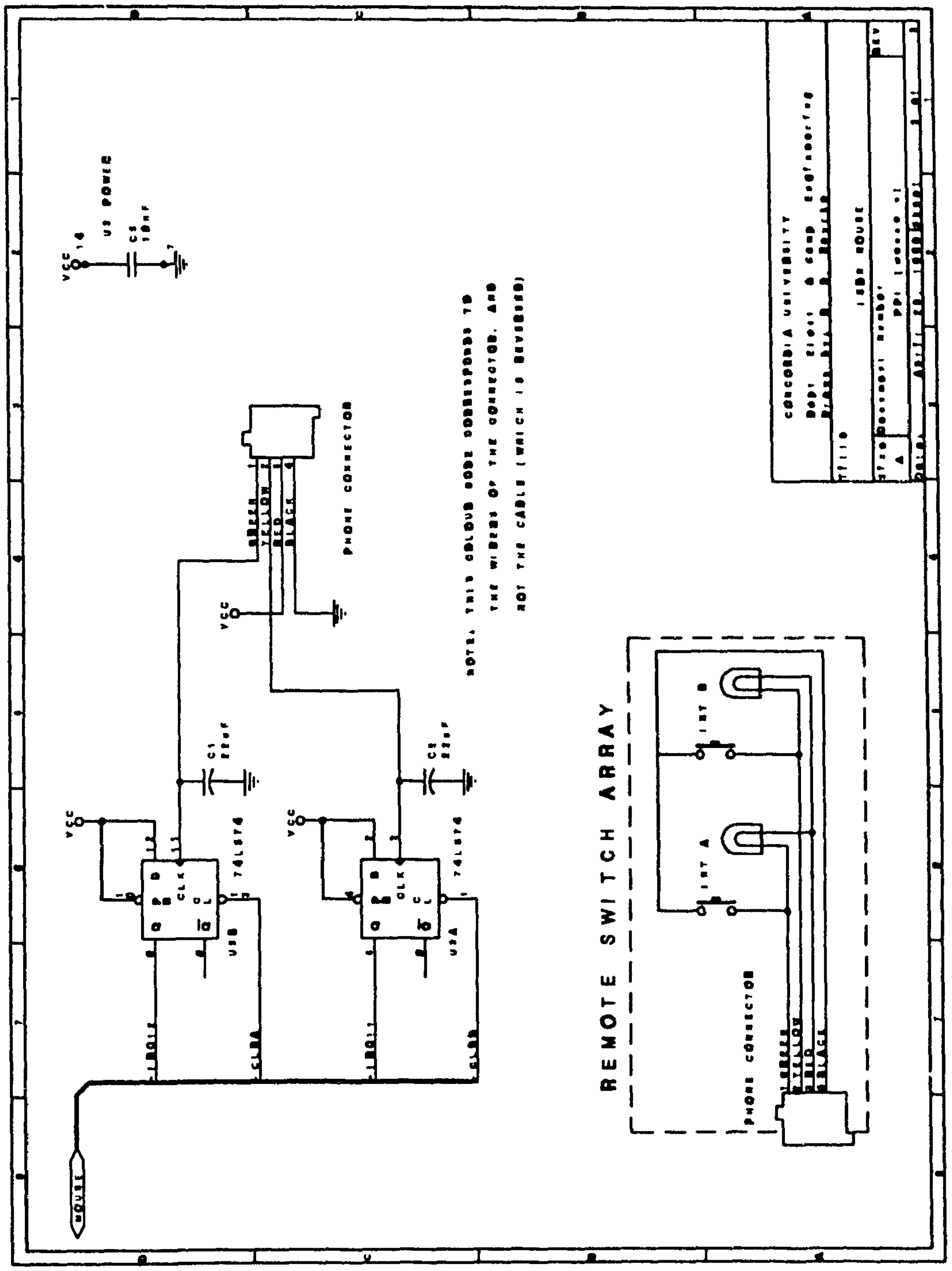
```
/* display message on console */
BYTEW = rq$s$write$move(co$conn,Msg1,strlen(Msg1),&status);
if (status != 0 )
      wrterr( status );

/* write message to file on the hard disk */
BYTEW = rq$s$write$move(hd$conn,Msg0,strlen(Msg0),&status);
if (status != 0 )
      wrterr( status );

inarray[0]=0;
while (inarray[0] != 'E') {

/* display prompt on console */
BYTEW = rq$s$write$move(co$conn,Msg2,strlen(Msg2),&status);
if (status != 0 )
      wrterr( status );

/* read user's console input */
byte$r = rq$s$read$move(ci$conn, inarray, 80, &status);
if (status != 0 )
      wrterr( status );

/* write user's console input to a logging file "hdcopy" */
BYTEW = rq$s$write$move(hd$conn, inarray , byte$r, &status);
if (status != 0 )
      wrterr( status );
BYTEW = rq$s$write$move(hd$conn,LF,2,&status);
if (status != 0 )
      wrterr( status );

/* write user's console input to the console output device */
BYTEW = rq$s$write$move(co$conn,Msg3,strlen(Msg3),&status);
if (status != 0 )
      wrterr( status );
BYTEW = rq$s$write$move(co$conn, inarray, byte$r, &status);
if (status != 0 )
      wrterr( status );
BYTEW = rq$s$write$move(co$conn,LF,2,&status);
if (status != 0 )
      wrterr( status );

/* get the time from the OS */
rq$get$global$time(&date,&status);
if (status != 0 )
      wrterr( status );
/* write the system date/time to a serial connected debugging terminal */
BYTEW = rq$s$write$move(co$comm,Msg5,strlen(Msg5),&status);
if (status != 0 )
      wrterr( status );
BYTEW = rq$s$write$move(co$comm, Hex2dec((int)date.years,msg),8, &status);
if (status != 0 )
      wrterr( status );
BYTEW = rq$s$write$move(co$comm, Hex2ascii((int)date.years,msg),8,
      &status);
if (status != 0 )
      wrterr( status );

Hex2dec((int)date.hours,datestr);
Hex2dec((int)date.minutes,&(datestr[5]));
Hex2dec((int)date.seconds,&(datestr[10]));
datestr[0] = '>';
datestr[5] = ':';
datestr[10] = ':';
BYTEW = rq$s$write$move(co$comm,Msg6,strlen(Msg6),&status);
if (status != 0 )
      wrterr( status );
BYTEW = rq$s$write$move(co$comm, datestr,18, &status);
if (status != 0 )
      wrterr( status );

k = 0;
while (k < 25) {
BYTEW = rq$s$write$move(co$conn, Hex2dec(k,msg),8, &status);
k = k+1;
}

} /*end while */

/* close connection for writting */
rq$s$close(hd$conn, &status);
if (status != 0 )
      wrterr( status );

/* open connection for input, 4 buffers */
rq$s$open(hd$conn, 1, 4, &status);
if (status != 0 )
      wrterr( status );

byte$r = 80;
while (byte$r == 80) {
      /* read message from file on the hard disk */
      byte$r = rq$s$read$move(hd$conn, inarray, 80, &status);
      if (status != 0 )
            wrterr( status );
      BYTEW = rq$s$write$move(co$conn, inarray, byte$r, &status);
      if (status != 0 )
            wrterr( status );
} /*end while */

/* close connection */
rq$s$close(hd$conn, &status);
if (status != 0 )
      wrterr( status );

rq$logical$detach$device(udistr(MessRMX,":CO:"), &status);
if (status != 0 )
      interr( status );

rq$logical$detach$device(udistr(MessRMX,":CC:"), &status);
if (status != 0 )
      interr( status );

rq$exit$io$job( 0,NULL, &status);
if (status != 0 )
      interr( status );

} /*end file*/
```

APPENDIX 3

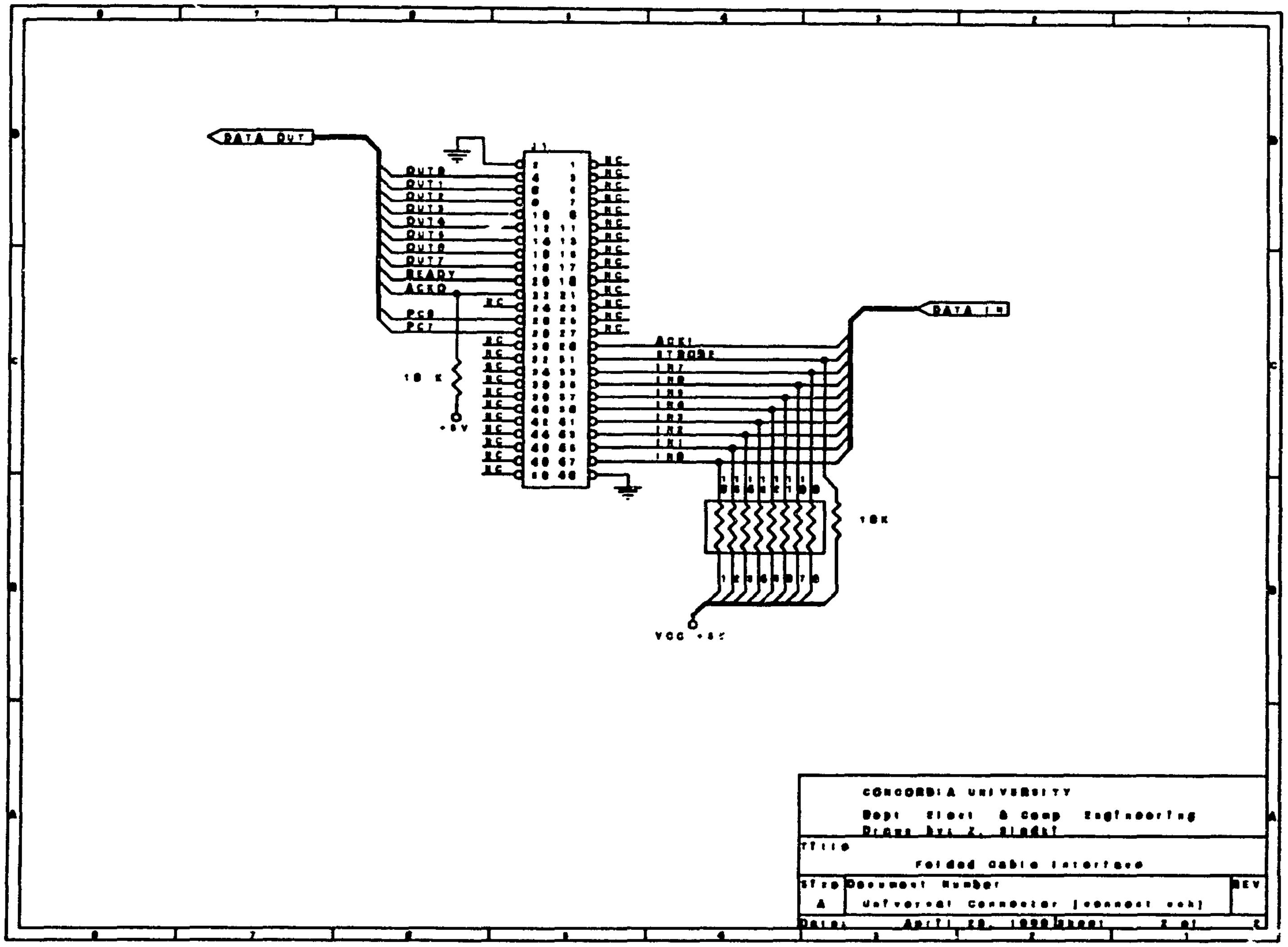
Schematic Diagrams: The PDLC and Interrupt Generator.

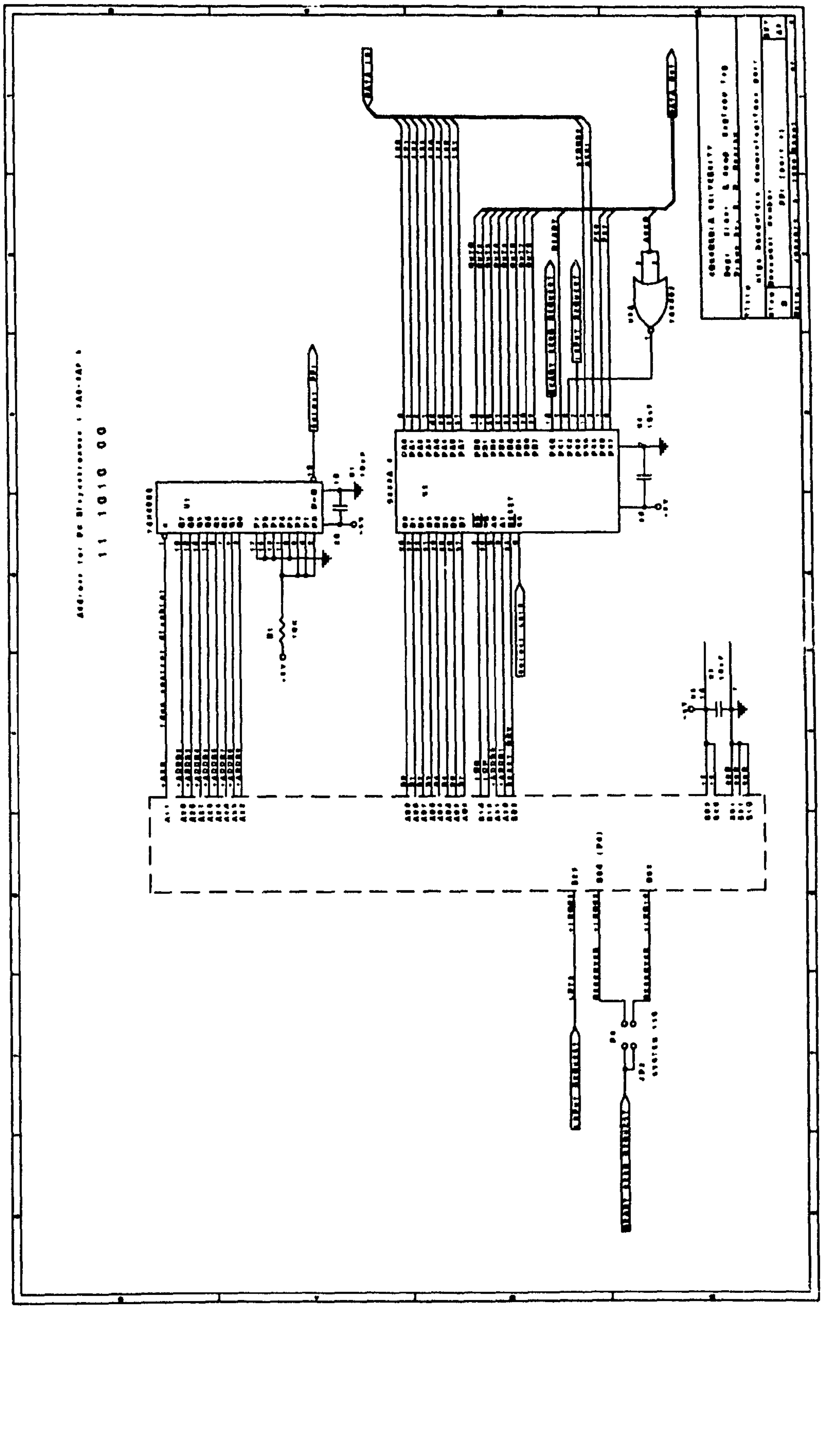
The schematic diagrams for the PDLC hardware and the interrupt generator (also called ISDN Mouse or HINTS), are included in this appendix. Functional description of these devices are provided in Chapter 4. The PDLC board used in the System 120 differs from the one in the IBM PC because it contains ISDN Mouse support. The ISDN Mouse consists of two switches that are connected to two workstation interrupt request lines. The control circuitry on the board permits either interrupt to be disabled through software. We built the ISDN Mouse to simulate externally connected hardware devices generating OS events.


11 1010 00
CONCORDIA UNIVERSITY
Dept. Elec. & Comp. Engineering
Title
Document Number
DATA OUT
DATA IN



VCC
14
U3 POWER
C3 10nF
7
PHONE CONNECTOR
GREEN
YELLOW
RED
BLACK
VCC
NOTE: THIS COLOUR CODE CORRESPONDS TO THE WIRES OF THE CONNECTOR, AND NOT THE CABLE (WHICH IS REVERSED)
C1 22nF
C2 22nF
74LS74
U3B
U3A
D
CLK
Q
CLRA
CLRB
MOUSE
REMOTE SWITCH ARRAY
INT A
INT B
PHONE CONNECTOR
GREEN
YELLOW
RED
BLACK
CONCORDIA UNIVERSITY
Dept. Elect. & Comp. Engineering
Title
ISDN MOUSE
Size
A
Document Number
REV
Date:


DATA OUT
DATA IN
J1
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
READY
ACKO
PC6
PC7
NC
ACKI
STROBE
IN7
IN6
IN5
IN4
IN3
IN2
IN1
IN0
10 K
+5V
10K
VCC +5V
CONCORDIA UNIVERSITY
Dept Elect & Comp Engineering
Drawn by: Z. Siddiqi
Title
Folded Cable Interface
Size
A
Document Number
Universal Connector (connect sch)
REV
Date: April 28, 1990
Sheet 2 of



APPENDIX 4

COSI token definitions

The COSI tokens that are used in our experimental implementation are listed in this appendix. These are the tokens through which the expert system accesses operating system services.

COSI TOKEN Description:

Classification: ES Tokens

Category: **General**

TOKEN 78
Function: Reply with a floating point value to an expert system enquiry.

Classification: ISDN Tokens

Category: **General**

TOKEN 1106
Function: Obtain and display ISDN connection status information.

TOKEN 1107
Function: Determine whether an autonomous voice call can be made by returning the ISDN connection status information for the voice channel. CRN = 0 signifies a free channel.

TOKEN 1108
Function: Determine whether an autonomous data call can be made by returning the ISDN connection status information for the data channel. CRN = 0 signifies a free channel.

TOKEN 1109
Function: Determine whether an autonomous file transfer call can be made by returning the ISDN connection status information for the file transfer channel. CRN = 0 signifies a free channel.

Category: **Voice Call**

TOKEN 1211
Function: Make an ISDN Voice Call.

TOKEN 1212
Function: Accept an incoming ISDN Voice Call.

TOKEN 1213
Function: Disconnect an ISDN Voice Call.

TOKEN 1214
Function: Reject an ISDN Voice Call.

TOKEN 1218
Function: Make an ISDN Voice Call and give full handset control to the user as well as the ability to disconnect the call and obtain ISDN connection status information.

TOKEN 1251
Function: Toggle handset microphone (ON/OFF).

TOKEN 1252
Function: Toggle handset earpiece (ON/OFF).

TOKEN 1253
Function: Increase handset earpiece volume.

TOKEN 1254
Function: Decrease handset earpiece volume.

Category: **Data Call**

TOKEN 1321
Function: Make an ISDN Data Call.

TOKEN 1322
Function: Accept an incoming ISDN Data Call.

TOKEN 1323
Function: Disconnect an ISDN Data Call.

TOKEN 1324
Function: Reject an ISDN Data Call.

TOKEN 1325
Function: Ask the user for a data string then transmit it.

TOKEN 1326
Function: Receive a data string and display it for the user.

TOKEN 1327
Function: Make an ISDN Data Call and ask the user for a data string then transmit it.

Category: **Keyboard Conversation Call**
(not implemented)

TOKEN 1431
Function: Make an ISDN Keyboard Conversation Call.

TOKEN 1432
Function: Accept an incoming ISDN Keyboard Conversation Call.

TOKEN 1433
Function: Disconnect an ISDN Keyboard Conversation Call.

TOKEN 1434
Function: Reject an ISDN Keyboard Conversation Call.

TOKEN 1435
Function: Enter the Keyboard Conversation mode.

TOKEN 1436
Function: Enter the Keyboard Conversation mode.

Category: **File Call**

TOKEN 1541
Function: Make an ISDN File Call.

TOKEN 1542
Function: Accept an incoming ISDN File Call.

TOKEN 1543
Function: Disconnect an ISDN File Call.

TOKEN 1544
Function: Reject an ISDN File Call.

TOKEN 1545
Function: Ask the user for a file name then transmit the file.

TOKEN 1546
Function: Ask the user for a file name then receive the file.

TOKEN 1547
Function: Make an ISDN File Call then transmit a default help file.

TOKEN 1548
Function: Make an ISDN File Call then transmit a default file used for person identification expert system.

APPENDIX 5

Program Listing: An Experimental ISDN Personal Workstation

The source code listing of the programs in our Experimental ISDN Personal Workstation are included in this appendix. The software is divided into three groups. The first group consists of the programs that run on the System 120 (pp.99-150). The second group consists of the programs that run on the IBM PC ISDN Server (pp.151-165). The last group consists of the programs which run on the IBM PC that simulate an ISDN NT terminal (pp.166-179).

System 120 Software Listing

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project      :    ISDN Personal Workstation                        =
   =    Sub-Project  :    RMX CONSTANTS                                    =
   =    File         :    rmxconst.h                                       =
   =    Author       :    Zenon Slodki                                     =
   =    Start Date   :    22 Mar 1990                                      =
   =    Update       :    11 Apr 1990                                      =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
/* classification of internal priority levels */
#define         LOW$PRIORITY      150
#define         MED$PRIORITY      100
#define         HIGH$PRIORITY     80
#define         SPUR$PRIORITY     50
#define         INTA$PRIORITY     44
#define         INTB$PRIORITY     42
#define         TRAN$PRIORITY     40
#define         RECV$PRIORITY     98

/* classification of internal interrupt levels */
#define         SPUR$LEVEL        0x27
#define         INTA$LEVEL        0x24
#define         INTB$LEVEL        0x23
#define         TRAN$LEVEL        0x22
#define         RECV$LEVEL        0x58

/* classification of mailbox types */
#define         DATA$TYPE         0x0020      /* FIFO Data mailbox */
#define         MSG$TYPE          0x0000      /* FIFO Message mailbox */

/* classification of semaphore types */
#define         FIFO              0           /* FIFO semaphore type */
#define         PRIORITY          1           /* Priority sem. type */
#define         NO$INITIAL        0           /*initially empty queue */
#define         SIZE$ONE          1           /* one unit size */
#define         ONE$UNIT          1           /* value of one unit */

/* General declarations */
#define         DATA$SEG          0           /* use own data segment */
#define         STACK$PTR         0L          /* auto. stack alloc. */
#define         STACK$SIZE        9000        /* bytes */
#define         NO$FLOATS         0           /* no flt.pt. instruc. */
#define         WITH$FLOATS       1           /* has flt.pt.instruc. */
#define         ONE$OUTSTAND      1           /* one unack. int */
#define         SELECT$OFNIL      0           /*refers to calling task*/
#define         LF                printf("\n")
#define         FALSE             0
#define         TRUE              1
#define         FOREVER           0xFFFF      /* temporal scope */
#define         NOWAIT            0           /* temporal scope */
#define         BURST$SIZE        100         /* max transmission size*/
#define         FILE_64K          20
#define         NIL               0

/* PDLC declarations */
#define         WRCH$BUF$SIZE     9000
#define         WTRH$BUF$SIZE     9000
#define         HEAD$SIZE         3
```

```
/* possible states of the global variable WRCT$MODE$GLB */
#define          READY$MODE          0
#define          BUFFER$MODE         1

/* possible header data classifications */
#define          STAT$REQ            0
#define          EXT$REQ             1
#define          BUF$REQ             2

/*
end file rmxconst.h */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :  ISDN Personal Workstation                       =
   =    Sub-Project   :  Initialization                                  =
   =    File          :  init.c                                          =
   =    Author        :  Zenon Slodki                                    =
   =    Start Date    :  21 Mar 1990                                     =
   =    Update        :  01 May 1990                                     =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#include  "/lib/cc286/stdio.h"
#include  "/lib/cc286/udi.h"
#include  "/lib/cc286/rmx.h"
#include  ":home:init/rmxconst.h"
#include  ":home:init/harts.hx"
#include  ":home:init/halos.hx"
#include  ":home:init/es.hx"
#include  ":home:init/cosi.hx"
#include  ":home:init/isdn.hx"
#include  ":home:pdlc/pdlc.hx"
extern      void  wait();

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function    :    main                                               -
 -     Input       :    none                                               -
 -     Output      :    none                                               -
 -     Action      :    Create all layers of the OS and interfaces         -
 -     Date        :    21 Mar 90                                          -
 -     UpDate      :    01 May 90                                          -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
main()
{
      word        status;

      harts$constructor();
      isdn$constructor();
      cosi$constructor();
      halos$constructor();
      pdlc$constructor();
      es$constructor();

      /* main task is no longer needed */
      rq$suspend$task(SELECT$OFNIL, &status);
}

void  wait()
{
      printf("\nPress Carriage Return to Continue.\n");
      getchar();
}

/*
end file init.c */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =   Project       :   ISDN Personal Workstation                  =
   =   Sub-Project   :   HARTS                                      =
   =   File          :   harts.h                                    =
   =   Author        :   Zenon Slodki                               =
   =   Start Date    :   09 Feb 1990                                =
   =   Update        :   23 Mar 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

/* function declarations */
void install();
void intatask();
void intbtask();
void spurtask();
void debounce$inta();
void debounce$intb();

/*
      end of file harts.h */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =   Project       :   ISDN Personal Workstation                  =
   =   Sub-Project   :   HARTS                                      =
   =   File          :   mouseint.hx                                =
   =   Author        :   Zenon Slodki                               =
   =   Start Date    :   13 Feb 1990                                =
   =   Update        :   23 Mar 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

/* 8255 PPI definitions */
#define      PORTA                  0x3A0
#define      PORTB                  0x3A1
#define      PORTC                  0x3A2
#define      PORT$CT                0x3A3
/*                                            76543210*/
#define      RUN$MODE               0xB4      /*10110100*/
#define      OUT$MODE               0x80      /*10000000*/
#define      INT$A                  9         /*00001001*/
#define      INT$B                  5         /*00000101*/
#define      SET$PC6                0x0D      /*00001101*/
#define      RST$PC6                0x0C      /*00001100*/
#define      SET$PC7                0x0F      /*00001111*/
#define      RST$PC7                0x0E      /*00001110*/

/* Assembler function declarations */
extern void     intahand();
extern void     intbhand();
extern void     spurhand();
extern void     output();

/*
      end of file mouseint.hx */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =   Project       :    ISDN Personal Workstation                    =
   =   Sub-Project   :    HARTS                                        =
   =   File          :    harts.c                                      =
   =   Author        :    Zenon Slodki                                 =
   =   Start Date    :    09 Feb 1990                                  =
   =   Update        :    01 May 1990                                  =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#include     "/lib/cc286/stdio.h"
#include     "/lib/cc286/udi.h"
#include     "/lib/cc286/rmx.h"
#include     "/user/zen/init/rmxconst.h"
#include     "harts.h"
#include     "mouseint.hx"


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :  harts$constructor                                 -
 -    Input       :  none                                              -
 -    Output      :  none                                              -
 -    Action      :  creates the HARTS layer task                      -
 -    Date        :  21 Mar 90                                         -
 -    UpDate      :  22 Mar 90                                         -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
      Supports hardware interrupts from the ISDN Mouse.
*/
/*   Function declarations */
extern void WHARTS$TASK();

void harts$constructor()
{
       word        status;

       /* create HARTS layer of the workstation operating system */
       rq$create$task(HIGH$PRIORITY, WHARTS$TASK, DATA$SEG, STACK$PTR,
                                 STACK$SIZE, NO$FLOATS, &status);

} /*end harts$constructor()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :  WHARTS$TASK                                       -
 -    Input       :  none                                              -
 -    Output      :  none                                              -
 -    Action      :  Create all interrupt tasks and semaphores         -
 -    Date        :  09 Feb 90                                         -
 -    UpDate      :  01 May 90                                         -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*   debouncing semaphore tokens for both hardware interrupts */
token        bounce$sem$A;
token        bounce$sem$B;

void WHARTS$TASK()
{
word         status;

       printf("\n HARTS initialization. \n");
/* create semaphores used to signal debouncing support routines */
bounce$sem$A = rq$create$semaphore(NO$INITIAL, SIZE$ONE, FIFO, &status);
bounce$sem$B = rq$create$semaphore(NO$INITIAL, SIZE$ONE, FIFO, &status);

/* create debouncing support routines */
     rq$create$task(LOW$PRIORITY, debounce$inta, DATA$SEG, STACK$PTR,
                              STACK$SIZE, NO$FLOATS, &status);
     rq$create$task(LOW$PRIORITY, debounce$intb, DATA$SEG, STACK$PTR,
                              STACK$SIZE, NO$FLOATS, &status);
/* program 8255 PPI */
     install();

/* create the interrupt handler for spurious interrupts (short signals) */
     rq$create$task(SPUR$PRIORITY, spurtask, DATA$SEG, STACK$PTR,
                              STACK$SIZE, NO$FLOATS, &status);

/* create the interrupt handler for ISDN Mouse switch A */
     rq$create$task(INTA$PRIORITY, intatask, DATA$SEG, STACK$PTR,
                              STACK$SIZE, NO$FLOATS, &status);

/* create the interrupt handler for ISDN Mouse switch B */
     rq$create$task(INTB$PRIORITY, intbtask, DATA$SEG, STACK$PTR,
                              STACK$SIZE, NO$FLOATS, &status);

     while( FOREVER ) {
          rq$suspend$task( SELECT$OFNIL, &status);
     } /*end while*/

} /*end WHARTS$TASK()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   intatask                               -
 -    Input       :   none                                   -
 -    Output      :   none                                   -
 -    Action      :   Mouse interrupt A support task         -
 -    Date        :   09 Feb 90                              -
 -    UpDate      :   22 Mar 90                              -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
    Supports ISDN Mouse Int A handler.
*/

void intatask()
{
     word              status;
     extern token      bounce$sem$A;
     extern token      WHALOS$COMMAND$DMB;

/*  Install the interrupt handler into the IDT and specify that this
    task will be the interrupt service task for the interrupt handler.
    Specify that no outstanding interrupt requests are allowed for
    the IRQ12 handler. */

rq$set$interrupt(INTA$LEVEL, ONE$OUTSTAND, intahand, DATA$SEG, &status);

     while( FOREVER ) {
          rq$wait$interrupt(INTA$LEVEL, &status);
          rq$send$units(bounce$sem$A, ONE$UNIT, &status);
          rq$send$data(WHALOS$COMMAND$DMB, "EVENT$A", 8, &status);
     } /*end while*/
} /*end intatask()*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :    intbtask                                   -
 -    Input       :    none                                       -
 -    Output      :    none                                       -
 -    Action      :    Mouse interrupt B support task             -
 -    Date        :    09 Feb 90                                  -
 -    UpDate      :    22 Mar 90                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
    Supports ISDN Mouse Int B handler.
*/

void intbtask()
{
     word              status;
     extern token      bounce$sem$B;
     extern token      WHALOS$COMMAND$DMB;


/*  Install the interrupt handler into the IDT and specify that this
     task will be the interrupt service task for the interrupt handler.
     Specify that no outstanding interrupt requests are allowed for
     the IRQ11 handler.
     */
rq$set$interrupt(INTB$LEVEL, ONE$OUTSTAND, intbhand, DATA$SEG, &status);

     while( FOREVER ) {
          rq$wait$interrupt(INTB$LEVEL, &status);
          rq$send$units(bounce$sem$B, ONE$UNIT, &status);
          rq$send$data(WHALOS$COMMAND$DMB, "EVENT$B", 8, &status);
     } /*end while*/
} /*end intbtask()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :    spurtask                                   -
 -    Input       :    none                                       -
 -    Output      :    none                                       -
 -    Action      :    Traps spurious interrupts on the slave PIC -
 -    Date        :    13 Feb 90                                  -
 -    UpDate      :    23 Mar 90                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
    Spurious interrupt handler for the slave PIC.
*/

void spurtask()
{
     word        status;

/*  Install the interrupt handler into the IDT and specify that this
     task will be the interrupt service task for the interrupt handler.
     Specify that no outstanding interrupt requests are allowed for
     the IRQ15 handler.
     */
rq$set$interrupt(SPUR$LEVEL, ONE$OUTSTAND, spurhand, DATA$SEG, &status);

     while( FOREVER ) {
          rq$sleep(200, &status);
          rq$wait$interrupt(SPUR$LEVEL. &status);
     } /*end while*/
} /*end spurtask()*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   install                                    -
 -    Input       :   none                                       -
 -    Output      :   none                                       -
 -    Action      :   initializes the 8255 PPI                   -
 -    Date        :   15 Jan 90                                  -
 -    UpDate      :   21 Mar 90                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* This procedure initializes the PPI to mode 1 and enables interrupts.
   However, the interrupt vector is not loaded and the PIC is not enabled.
*/
void install()
{

/*  set PPI to mode 1 */
      output(PORT$CT, RUN$MODE);

/*  enable interrupts */
      output(PORT$CT, INT$A);
      output(PORT$CT, INT$B);

/*  initialize debounce circuitry for Int B */
      output(PORT$CT, RST$PC6);
      output(PORT$CT, SET$PC6);

/*  initialize debounce circuitry for Int A */
      output(PORT$CT, RST$PC7);
      output(PORT$CT, SET$PC7);

} /*end install()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   debounce$inta                              -
 -    Input       :   none                                       -
 -    Output      :   none                                       -
 -    Action      :   debounces interrupt A                      -
 -    Date        :   16 Feb 90                                  -
 -    UpDate      :   05 Apr 90                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*   This task renables the flip-flop for IRQ12 (interrupt A).
*/
void debounce$inta()
{
      word              status;
      extern token      bounce$sem$A;

      while( FOREVER ) {

/*          printf("\aInt A Ready\n"); */
            rq$receive$units(bounce$sem$A, ONE$UNIT, FOREVER, &status);
            rq$sleep(100, &status);
            output(PORT$CT, SET$PC7);
      } /*end while*/

} /*end debounce$inta()*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function   :   debounce$intb                                  -
 -    Input      :   none                                           -
 -    Output     :   none                                           -
 -    Action     :   debounces interrupt B                          -
 -    Date       :   16 Feb 90                                      -
 -    UpDate     :   05 Apr 90                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/*   This task renables the flip-flop for IRQ11 (interrupt B).
*/
void debounce$intb()
{
      word              status;
      extern token      bounce$sem$B;

      while( FOREVER ) {
/*          printf("\aInt B Ready\n"); */
            rq$receive$units(bounce$sem$B, ONE$UNIT, FOREVER, &status);
            rq$sleep(100, &status);
            output(PORT$CT, SET$PC6);
      } /*end while*/
} /*end debounce$intb()*/

/*
      end file harts.c */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   ISDN Personal Workstation               =
   =    Sub-Project   :   Parallel Communication Protocol         =
   =    File          :   pdlc.c                                  =
   =    Author        :   Zenon Slodki                            =
   =    Start Date    :   28 Mar 1990                             =
   =    Update        :   01 May 1990                             =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#include  "/lib/cc286/stdio.h"
#include  "/lib/cc286/udi.h"
#include  "/lib/cc286/rmx.h"
#include  "pdlcint.hx"
#include  "/user/zen/harts/mouseint.hx"
#include  "/user/zen/init/rmxconst.h"


/* Used to test custom parallel communication boards developed by ZenRob
   in the interrupt mode of operation.
*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function   :   pdlc$constructor                              -
 -    Input      :   none                                          -
 -    Output     :   none                                          -
 -    Action     :   Create all interrupt tasks and mailboxes      -
 -    Date       :   28 Mar 90                                     -
 -    UpDate     :   01 May 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* mailbox token to transmitter task interface */
token WTRT$COMMAND$DMB;

/* Function declarations */
extern void WTRT$TASK();
extern void WRCT$TASK();

void pdlc$constructor()
{
      extern token      WTRT$COMMAND$DMB;
      word              status;

      wait();
      printf("PDLC constructor\n");

      /* create interface to WTRT$TASK */
      WTRT$COMMAND$DMB = rq$create$mailbox(DATA$TYPE, &status);

      /* create transmitter task */
      rq$create$task(TRAN$PRIORITY, WTRT$TASK, DATA$SEG, STACK$PTR,
                                STACK$SIZE, NO$FLOATS, &status);

      /* create receiver task */
      rq$create$task(RECV$PRIORITY, WRCT$TASK, DATA$SEG, STACK$PTR,
                                STACK$SIZE, NO$FLOATS, &status);

} /*end pdlc$constructor()*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function   :   WRCT$TASK                                     -
 -    Input      :   none                                          -
 -    Output     :   none                                          -
 -    Action     :   Supports Interrupt Handler for the receiver   -
 -    Date       :   28 Mar 90                                     -
 -    UpDate     :   11 Apr 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* This task supports the receiver interrupt handler. It first installs
   the handler and then performs the data processing whenever the handler
   signals it that it has data.
*/
void WRCT$TASK()
{
        extern word         WRCT$MODE$GLB;
        extern char         WRCH$HEAD$GLB[3];
        extern token        WISDN$RESPONSE$DMB;
        extern token        WHALOS$COMMAND$DMB;
        extern word         WRCT$BUF$EOB;
        extern word         WRCH$BUF$IN;
        extern word         WRCT$COUNT$GLB;
        word                recv$size;
        word                status;

        /*  Install the interrupt handler into the IDT and specify that this
               task is the interrupt service task for the interrupt handler.
               Specify that no outstanding interrupt requests are allowed for
               the IRQ5 handler.
        */

        WRCT$MODE$GLB = READY$MODE;

rq$set$interrupt(RECV$LEVEL,ONE$OUTSTAND, WRCH$TASK, DATA$SEG, &status);

        while( FOREVER ) {
        /* wait for the interrupt handler to signal you that a character is
           in the buffer.
        */
                WRCT$COUNT$GLB = 0;          /* reset header pointer */

                rq$wait$interrupt(RECV$LEVEL, &status);

#ifdef TESTING
                printf("WRCH$HEAD$GLB: %u, %u, %u \n",WRCH$HEAD$GLB[0],
                WRCH$HEAD$GLB[1],WRCH$HEAD$GLB[2]);
#endif

                switch( WRCH$HEAD$GLB[0] ) {
                        /* i.e. response (status) to a previous command */
                        case STAT$REQ:
                                rq$send$data(WISDN$RESPONSE$DMB, WRCH$HEAD$GLB,
                                                        HEAD$SIZE, &status);
                                break;

                        /* external request processing */
                        case EXT$REQ:
                                rq$send$data(WHALOS$COMMAND$DMB, WRCH$HEAD$GLB,
                                                        HEAD$SIZE, &status);
                                break;
```

```
                  /* buffer processing */
                  case BUF$REQ:
                        /* access bytes 2 and 3 */
                        recv$size = * (int*) &WRCH$HEAD$GLB[1];
#'fdef TESTING
                        printf("BUF$REQ: recv$size %u \n", recv$size);
#endif
                        WRCT$BUF$EOB = (WRCH$BUF$IN + recv$size)
                                          % WRCH$BUF$SIZE;

                        WRCT$MODE$GLB = BUFFER$MODE;
                        rq$wait$interrupt(RECV$LEVEL, &status);
                        WRCT$MODE$GLB = READY$MODE;

                        rq$send$data(WISDN$RESPONSE$DMB, WRCH$HEAD$GLB,
                                          HEAD$SIZE, &status);
                        break;

                  default:
                        printf("WRCT$TASK: unknown request -> %u \n",
                              WRCH$HEAD$GLB[0]);

                  } /*end switch*/
      } /*end while*/
} /*end WRCT$TASK()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :    WTRT$TASK                                   -
 -    Input       :    none                                        -
 -    Output      :    none                                        -
 -    Action      :    Supports Interrupt Handler for the transmitter-
 -    Date        :    28 Mar 90                                   -
 -    UpDate      :    11 Apr 90                                   -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* This task supports the transmitter interrupt handler. It first installs
   the handler and then performs the data processing whenever the handler
      signals it that it no longer has data.
*/
word  msg$size;

void WTRT$TASK()
{
extern byte WTRH$BUF$RING[WTRH$BUF$SIZE];
      extern token      WTRT$COMMAND$DMB;
      extern word       WTRT$BUF$EOB;
      extern word       WTRH$BUF$OUT;
      word              status;

      /*  Install the interrupt handler into the IDT and specify that this
            task is the interrupt service task for the interrupt handler.
            Specify that no outstanding interrupt requests are allowed for
            the IRQ5 handler.
      */

rq$set$interrupt(TRAN$LEVEL,ONE$OUTSTAND, WTRH$TASK, DATA$SEG, &status);

      while( FOREVER ) {

      /* wait for a transmission request */
      rq$receive$data(WTRT$COMMAND$DMB, &msg$size, FOREVER, &status);
```

```
              WTRT$BUF$EOB = (WTRH$BUF$OUT + msg$size) % WTRH$BUF$SIZE;

#ifdef TESTING
              printf(&WTRH$BUF$RING[WTRH$BUF$OUT]);
#endif

         /* wait for the interrupt handler to signal you that the data buffer
              has reached the EOB marker */
              rq$wait$interrupt(TRAN$LEVEL, &status);

         }  /*end while*/
} /* end WTRT$TASK */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   error_hand                                  -
-    Input            :   error_code, procedure seg & offset          -
-    Output           :   none                                        -
-    Date             :   24 Jan 90                                   -
-    UpDate           :   26 Jan 90                                   -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* This procedure displays the error condition and the segment and offset
   of the procedure in trouble.
*/
void error_hand(error_code, proc_seg, proc_off)
int error_code;
int proc_seg;
int proc_off;
{
char temp;
         printf("ERROR HANDLER CODE: %u\n", error_code);
         printf("CS: %u, IP: %u\n", proc_seg, proc_off);
         dq$exit( error_code );
         scanf("%c",&temp);            /* indefinite wait */
} /* end error_hand */

/*
end file pdlc.c */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   ISDN Personal Workstation                      =
   =    Sub-Project   :   HALOS                                          =
   =    File          :   halos.c                                        =
   =    Author        :   Zenon Slodki                                   =
   =    Start Date    :   22 Mar 1990                                    =
   =    Update        :   01 May 1990                                    =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#include  "/lib/cc286/stdio.h"
#include  "/lib/cc286/udi.h"
#include  "/lib/cc286/rmx.h"
#include  "/user/zen/init/rmxconst.h"


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function     :   halos$constructor                                -
 -     Input        :   ncne                                             -
 -     Output       :   none                                             -
 -     Action       :   creates the HALOS layer task                     -
 -     Date         :   21 Mar 90                                        -
 -     UpDate       :   23 Mar 90                                        -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
     Event-driven application schedular.
*/
/*    Function declarations */
extern void       WHALOS$TASK();

/*   interface mailbox to HALOS for EXTERNAL requests */
token       WHALOS$COMMAND$DMB;

void halos$constructor()
{
      extern      token WHALOS$COMMAND$DMB;
      word        status;

      /* create interface to HALOS layer */
      WHALOS$COMMAND$DMB = rq$create$mailbox(DATA$TYPE, &status);

      /* create HALOS layer of the workstation operating system */
      rq$create$task(LOW$PRIORITY, WHALOS$TASK, DATA$SEG, STACK$PTR,
                              STACK$SIZE, NO$FLOATS, &status);

} /*end halos$constructor()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function     :   WHALOS$TASK                                      -
 -     Input        :   none                                             -
 -     Output       :   none                                             -
 -     Action       :   Schedules applications                           -
 -     Date         :   22 Mar 90                                        -
 -     UpDate       :   01 May 90                                        -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
char              event[30];
void WHALOS$TASK()
{
      word              num$bytes$recv;
      word              status;
      extern token      WHALOS$COMMAND$DMB;
```

```
        extern token        WES$COMMAND$DMB;
        word                one=1;
        word                two=2;

        while( FOREVER ) {
              num$bytes$recv =
              rq$receive$data(WHALOS$COMMAND$DMB,&event[0],FOREVER,&status);

              if( num$bytes$recv != 8 )
              printf("HALOS: Mailbox received -> %d \n", num$bytes$recv);

              switch( event[6] ) {
                    case 'A':
                          rq$send$data(WES$COMMAND$DMB, &one, 2, &status);
                          break;
                    case 'B':
                          rq$send$data(WES$COMMAND$DMB, &two, 2, &status);
                          break;
                    default:
                          printf("HALOS: unknown token -> %c \n", event[6]);
              } /*end switch*/

        } /*end while*/
  } /*end WHALOS$TASK()*/

  /*
  end file halos.c */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :  ISDN Personal Workstation                    =
   =    Sub-Project   :  COSI                                         =
   =    File          :  cosi.c                                       =
   =    Author        :  Zenon Slodki                                 =
   =    Start Date    :  22 Mar 1990                                  =
   =    Update        :  01 May 1990                                  =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/
#define      RMXON
#include  "/lib/cc286/stdio.h"
#include  "/lib/cc286/udi.h"
#include  "/lib/cc286/rmx.h"
#include  "/user/zen/init/rmxconst.h"
#include  "/user/robert/koola/hi.hx"
#include  "/user/robert/koola/huminter.h"

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   cosi$constructor                                -
 -    Input       :   none                                            -
 -    Output      :   none                                            -
 -    Action      :   creates the COSI layer task                     -
 -    Date        :   21 Mar 90                                       -
 -    UpDate      :   01 May 90                                       -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
     Centralized, abstract interface primitives provider. */

/*    Function declarations */
extern void      WCOSI$TASK();

/*   interface mailbox to COSI for ACTION and REQUESTS tokens */
token       WCOSI$COMMAND$DMB;

/*   interface mailbox to COSI for floating point replies */
token       WCOSI$RESPONSE$DMB;

/*  semaphore indicates when the ES can continue processing after sending
      a command to COSI */
token       WCOSI$READY$1US;

void cosi$constructor()
{
      extern token      WCOSI$COMMAND$DMB;
      word              status;

      /* create command/request interface to COSI layer */
      WCOSI$COMMAND$DMB = rq$create$mailbox(DATA$TYPE, &status);

      /* create response interface to COSI layer */
      WCOSI$RESPONSE$DMB = rq$create$mailbox(DATA$TYPE, &status);

      /* create control access to ES layer */
      WCOSI$READY$1US = rq$create$semaphore(NO$INITIAL, ONE$UNIT, FIFO,
                        &status);

      /* create COSI layer of the workstation operating system */
      rq$create$task(LOW$PRIORITY, WCOSI$TASK, DATA$SEG, STACK$PTR,
                              STACK$SIZE, WITH$FLOATS, &status);

} /*end cosi$constructor()*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   WCOSI$TASK                                  -
 -    Input       :   none                                        -
 -    Output      :   none                                        -
 -    Action      :   Provides abstract interfaces.               -
 -    Date        :   22 Mar 90                                   -
 -    UpDate      :   01 May 90                                   -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
word            action;
word            rep_stat;
void WCOSI$TASK()
{
      word              num$bytes$recv;
      word              status;
      word              tok$val;
      extern token      WCOSI$COMMAND$DMB;
      extern token      WISDN$COMMAND$DMB;
      extern token      WISDN$STATUS$DMB;
      float             answer;
      word              reply=0;
      extern word       crn_voice;
      extern word       crn_data;
      extern word       crn_file;

      while( FOREVER ) {
      num$bytes$recv =
      rq$receive$data(WCOSI$COMMAND$DMB, &action, FOREVER, &status);
            if( num$bytes$recv != 2 )
            printf("COSI: Mailbox received -> %d \n", num$bytes$recv);
#ifdef TESTING
            printf("ACTION: %u \n", action);
#endif
            switch( action ) {

                  /* enquiry to COSI */
                  case 78:
                  /* reply to the expert system */
                  answer = 79.8;
                  printf("External request from ES token: 79.8\n");
                  rq$send$data(WCOSI$RESPONSE$DMB, &answer, 4, &status);
                  break;

                  case 1107:
                  /* obtain Voice call connection status information  */
                  tok$val = 07;
                  rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                  rq$sleep(120, &status);
                  answer = (float) crn_voice;
                  rq$send$data(WCOSI$RESPONSE$DMB, &answer, 4, &status);
                  break;

                  case 1108:
                  /* obtain Data call connection status information  */
                  tok$val = 07;
                  rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                  rq$sleep(120, &status);
                  answer = (float) crn_data;
                  rq$send$data(WCOSI$RESPONSE$DMB, &answer, 4, &status);
                  break;
```

```
case 1109:
/* obtain File call connection status information  */
tok$val = 07;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
rq$sleep(120, &status);
answer = (float) crn_file;
rq$send$data(WCOSI$RESPONSE$DMB, &answer, 4, &status);
break;


/* action to COSI */
case 1:
/* action taken by COSI is to shut down system */
printf("About to shut down system.\n");
printf("\n\a\aACTION taken: gracefull EXIT\n");
tok$val = 0;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

case 101:
/* action taken by COSI */
printf("TANK identified: T54/T55\n");
printf("\n\a\aACTION taken: ENGAGED with HESH\n");
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

case 102:
/* action taken by COSI */
printf("TANK identified: T62\n");
printf("\n\a\aACTION taken: ENGAGED with APFSDS\n");
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

case 103:
/* action taken by COSI */
printf("TANK identified: T80\n");
printf("\n\a\aACTION taken: RETREATE\n");
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

case 151:
/* action taken by COSI */
printf("TANK identified: LEOPARD\n");
printf("\n\a\aNO ACTION taken\n");
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

/* general ISDN services */
case 1106:
/* obtain ISDN status information -v- */
tok$val = 06;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
rq$sleep(200, &status);
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;
```

```
/* ISDN Voice services */
case 1211:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* make a voice call -a- */
tok$val = 11;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1212:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* accept a voice call */
tok$val = 12;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1213:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* disconnect a voice call -b- */
tok$val = 13;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1214:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* reject a voice call -c- */
tok$val = 14;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1218:
ask_quest( HIQST_PHONE_NUMBERS );
/* make a voice call -a- */
tok$val = 11;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
while( reply != 13 ) {
  switch( (int) ask_quest( HIQST_VOICE_ACTION ) ) {
  case 1:
     /* disconnect a voice call -b- */
     reply = tok$val = 13;
     rq$send$data(WISDN$COMMAND$DMB,&tok$val,2,&status);
     break;
  case 2:
     /* adjust increase volume -u- */
     tok$val = 53;
     rq$send$data(WISDN$COMMAND$DMB,&tok$val,2,&status);
     break;
  case 3:
     /* adjust decrease volume -u- */
     tok$val = 54;
     rq$send$data(WISDN$COMMAND$DMB,&tok$val,2,&status);
     break;
  case 4:
     /* toggle earpiece -t- */
     tok$val = 52;
     rq$send$data(WISDN$COMMAND$DMB,&tok$val,2,&status);
     break;
  case 5:
     /* toggle microphone -s- */
     tok$val = 51;
     rq$send$data(WISDN$COMMAND$DMB,&tok$val,2,&status);
     break;
  case 6:
     /* obtain ISDN status information -v- */
     tok$val = 06;
     rq$send$data(WISDN$COMMAND$DMB,&tok$val,2,&status);
     rq$sleep(200, &status);
     break;
  default:
     printf("Unknown reply \n");
     break;

  } /*end switch*/
  rq$sleep(60, &status);
} /*end while*/
reply = 0;
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

case 1251:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* toggle microphone -s- */
tok$val = 51;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1252:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* toggle earpiece -t- */
tok$val = 52;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1253:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* adjust increase volume -u- */
tok$val = 53;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1254:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
/* adjust decrease volume -u- */
tok$val = 54;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;
```

```
/* ISDN Data services */
case 1321:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* make a data call -d- */
tok$val = 21;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1322:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* accept a data call */
tok$val = 22;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1323:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* disconnect a data call -e- */
tok$val = 23;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
rq$receive$data(WISDN$STATUS$DMB, &rep_stat, FOREVER,
                      &status);
break;

case 1324:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* reject a data call -f- */
tok$val = 24;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1325:
/* send data string -g- */
tok$val = 25;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
rq$sleep( 1000, &status );

/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

case 1326:
/* receive data string -h- */
tok$val = 26;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
rq$sleep( 1000, &status );

/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;
```

```
case 1327:
/* automatically send a data string */
/* make a data call -d- */
tok$val = 21;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);

/* send data string -g- */
tok$val = 25;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
rq$sleep( 800, &status );

/* disconnect a data call -e- */
tok$val = 23;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);

rq$receive$data(WISDN$STATUS$DMB,  &rep_stat,  FOREVER,
                   &status);
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;


/* ISDN Keyboard Conversation services */
case 1431:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* make a keyboard call -i- */
tok$val = 31;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1432:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* accept a keyboard call */
tok$val = 32;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1433:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* disconnect a keyboard call -j- */
tok$val = 33;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1434:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* reject a keyboard call -k- */
tok$val = 34;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;
```

```
case 1435:
/* keyboard talk -l- */
tok$val = 35;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);

/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;

case 1436:
/* keyboard talk -m- */
tok$val = 36;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);

/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
break;


/* ISDN File services */
case 1541:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* make a file call -n- */
tok$val = 41;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1542:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* accept a file call */
tok$val = 42;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1543:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* disconnect a file call -o- */
tok$val = 43;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1544:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

/* reject a file call -p- */
tok$val = 44;
rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
break;

case 1545:
/* let the expert system continue processing */
rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
```

```
                    /* file transmit -q- */
                    tok$val = 45;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                    break;

                    case 1546:
                    /* let the expert system continue processing */
                    rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

                    /* file receive -r- */
                    tok$val = 46;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                    break;

                    case 1547:
                    /* let the expert system continue processing */
                    rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

                    /* make a file call -n- */
                    tok$val = 41;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);

                    /* predefined file transmit -q- */
                    tok$val = 48;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                    rq$sleep( 1200, &status );

                    /* disconnect a file call -o- */
                    tok$val = 43;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                    break;

                    case 1548:
                    /* let the expert system continue processing */
                    rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);

                    /* make a file call -n- */
                    tok$val = 41;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);

                    /* predefined file transmit -q- */
                    tok$val = 49;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                    rq$sleep( 1200, &status );

                    /* disconnect a file call -o- */
                    tok$val = 43;
                    rq$send$data(WISDN$COMMAND$DMB, &tok$val, 2, &status);
                    break;


                    default:
                    printf("COSI: unknown token -> %u \n", action);
                    /* let the expert system continue processing */
                    rq$send$units(WCOSI$READY$1US, ONE$UNIT, &status);
                } /*end switch*/

        } /*end while*/
} /*end WCOSI$TASK()*/

/*
end file cosi.c */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :   ISDN Personal Workstation                    =
   =    Sub-Project   :   ISDN                                         =
   =    File          :   isdn.h                                       =
   =    Author        :   Zenon Slodki                                 =
   =    Start Date    :   22 Mar 1990                                  =
   =    Update        :   12 Apr 1990                                  =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

/* function declarations */

int    call_isdn_dial();
void   call_isdn_voice_accept();
int    call_isdn_dial_disconnect();
int    call_isdn_voice_reject();
int    call_isdn_connect();
int    call_isdn_connect_accept();
int    call_isdn_connect_reject();
int    call_isdn_disconnect();
void   send_isdn_transmit();
void   call_isdn_transmit();
void   call_file_transmit();
void   send_file_transmit();
void   call_isdn_receive_wait();
void   send_isdn_receive();
void   call_file_receive_wait();
void   send_file_receive();
void   call_isdn_audio_vol();
void   call_isdn_mute_mic();
void   call_isdn_mute_ear();
int    Send_Command();
void   display_file();
void   call_isdn_status();
void   get_isdn_status();
/*
end file isdn.h */
```

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :  ISDN Personal Workstation                  =
   =    Sub-Project   :  ISDN                                       =
   =    File          :  isdn.c                                     =
   =    Author        :  Zenon Slodki                               =
   =    Start Date    :  22 Mar 1990                                =
   =    Update        :  01 May 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

#include  "/lib/cc286/stdio.h"
#include  "/lib/cc286/udi.h"
#include  "/lib/cc286/rmx.h"
#include  "isdn.h"
#include  "/user/zen/init/rmxconst.h"
#include  "/user/zen/harts/harts.h"
#include  "/user/zen/harts/mouseint.hx"
#include  "app.h"
#include  "/user/zen/pdlc/pdlcint.hx"

/* Global variables */
int  crn_voice = 0;
int  crn_data = 0;
int  crn_second = 0;
int  crn_file = 0;
int  main_choice = 0;
byte bfr[BURST$SIZE+5];
byte auto$file = 0;

/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
   =    Project       :  ISDN Personal Workstation                  =
   =    Sub-Project   :  Parallel Communication Protocol            =
   =    File          :  ring.c                                     =
   =    Author        :  Zenon Slodki                               =
   =    Start Date    :  05 Apr 1990                                =
   =    Update        :  10 Apr 1990                                =
   = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :  Send$Byte                                      -
 -    Input       :  data byte                                      -
 -    Output      :  none                                           -
 -    Action      :  Prepares the data for transmission on the PDLC-
 -    Date        :  05 Apr 90                                      -
 -    UpDate      :  11 Apr 90                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* This function places one byte in the transmitter interrupt handler's
   ring buffer and updates the output pointer.
   After this, the number of bytes of data to be transmitted is mailed to
   the transmitter interrupt task.
*/
word  one=1;
void Send$Byte( data$byte )
byte  data$byte;
{
      extern byte       WTRH$BUF$RING[WTRH$BUF$SIZE];
      extern token      WTRT$COMMAND$DMB;
      extern word       WTRH$BUF$IN;
      word              status;

      WTRH$BUF$RING[WTRH$BUF$IN] = data$byte;
```

```
       /* wrap around index */
       WTRH$BUF$IN +=1;
       if( WTRH$BUF$IN == WTRH$BUF$SIZE )
             WTRH$BUF$IN = 0;

       /* send size of data to be sent to the transmitter interrupt task */
       rq$send$data(WTRT$COMMAND$DMB, &one, 2, &status);
} /* end Send$Byte */


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function    :   Send$Word                                      -
 -     Input       :   data word                                      -
 -     Output      :   none                                           -
 -     Action      :   Prepares the data for transmission on the PDLC-
 -     Date        :   05 Apr 90                                      -
 -     UpDate      :   11 Apr 90                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* This function places one word in the transmitter interrupt handler's
   ring buffer and updates the output pointer.
   After this, the number of bytes of data to be transmitted is mailed to
   the transmitter interrupt task.
*/
word  two=2;
void Send$Word( data$word )
word  data$word;
{
       extern byte       WTRH$BUF$RING[WTRH$BUF$SIZE];
       extern token      WTRT$COMMAND$DMB;
       extern word       WTRH$BUF$IN;
       byte              *data$byte;
       word              status;

       data$byte = (char*) &data$word;

       /* load LSB data */
       WTRH$BUF$RING[WTRH$BUF$IN] = data$byte[0];
       /* wrap around index */
       WTRH$BUF$IN +=1;
       if( WTRH$BUF$IN == WTRH$BUF$SIZE )
             WTRH$BUF$IN = 0;

       /* load MSB data */
       WTRH$BUF$RING[WTRH$BUF$IN] = data$byte[1];
       /* wrap around index */
       WTRH$BUF$IN +=1;
       if( WTRH$BUF$IN == WTRH$BUF$SIZE )
             WTRH$BUF$IN = 0;

       /* send size of data to be sent to the transmitter interrupt task */
       rq$send$data(WTRT$COMMAND$DMB, &two, 2, &status);
} /* end Send$Word */


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function    :   Send$Buf                                       -
 -     Input       :   data buffer pointer                            -
 -     Output      :   none                                           -
 -     Action      :   Prepares the data for transmission on the PDLC-
 -     Date        :   05 Apr 90                                      -
 -     UpDate      :   09 Apr 90                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
/* This function places a string in the transmitter interrupt handler's
   ring buffer and updates the output pointer.
   After this, the number of bytes of data to be transmitted is mailed to
   the transmitter interrupt task.*/

void Send$Buf( data$buf )
byte  *data$buf;
{
      extern byte       WTRH$BUF$RING[WTRH$BUF$SIZE];
      extern token      WTRT$COMMAND$DMB;
      extern word       WTRH$BUF$IN;
      word              buf$size;
      word              raw$size,i;
      word              status;

      buf$size = strlen(data$buf) + 1; /* does not count the \0 */

      if( (raw$size = buf$size + WTRH$BUF$IN) < WTRH$BUF$SIZE ) {
            strncpy(&WTRH$BUF$RING[WTRH$BUF$IN], data$buf, buf$size);
            WTRH$BUF$IN = raw$size;
      }
      else
            /* wrap around index */
            for(i=0; i<buf$size; i++)  {
                  WTRH$BUF$RING[WTRH$BUF$IN] = data$buf[i];
                  WTRH$BUF$IN +=1;
                  if( WTRH$BUF$IN == WTRH$BUF$SIZE )
                        WTRH$BUF$IN = 0;
            }

      /* send size of data to be sent to the transmitter interrupt task */
      rq$send$data(WTRT$COMMAND$DMB, &buf$size, 2, &status);
} /* end Send$Buf */


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   Send$Num$Bytes                                -
 -    Input       :   data buffer pointer, data size                -
 -    Output      :   none                                          -
 -    Action      :   Prepares the data for transmission on the PDLC-
 -    Date        :   07 Apr 90                                     -
 -    UpDate      :   16 Apr 90                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* This function places a number of bytes of data into the transmitter
   interrupt handler's ring buffer and updates the output pointer.
   After this, the number of bytes of data to be transmitted is mailed to
   the transmitter interrupt task.
*/

void Send$Num$Bytes( data$buf, data$size )
byte  *data$buf;
word  data$size;
{
      extern byte       WTRH$BUF$RING[WTRH$BUF$SIZE];
      extern token      WTRT$COMMAND$DMB;
      extern word       WTRH$BUF$IN;
      word              raw$size,i;
      word              status;

      if( (raw$size = data$size + WTRH$BUF$IN) < WTRH$BUF$SIZE ) {
            strncpy(&WTRH$BUF$RING[WTRH$BUF$IN], data$buf, data$size);
            WTRH$BUF$IN = raw$size;
```

```
        }
        else
              /* wrap around index */
              for(i=0; i<data$size; i++) {
                    WTRH$BUF$RING[WTRH$BUF$IN] = data$buf[i];
                    WTRH$BUF$IN +=1;
                    if( WTRH$BUF$IN == WTRH$BUF$SIZE )
                          WTRH$BUF$IN = 0;
              }

#ifdef TESTING
        printf("data$size: %d, WTRH$BUF$IN: %d\n", data$size, WTRH$BUF$IN);
#endif

        /* send size of data to be sent to the transmitter interrupt task */
        rq$send$data(WTRT$COMMAND$DMB, &data$size, 2, &status);
/*      rq$sleep(120, &status); */

#ifdef TESTING
        printf("RING: bfr sent\n");
#endif

} /* end Send$Num$Bytes */

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -      Function    :   Display$Buf                                 -
 -      Input       :   none                                        -
 -      Output      :   none                                        -
 -      Action      :   Displays the data received in the ring buffer -
 -      Date        :   06 Apr 90                                   -
 -      UpDate      :   11 Apr 90                                   -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* This function displays the latest buffer data that is in the receiver
   ring buffer.
*/

void Display$Buf()
{
        extern byte       WRCH$BUF$RING[WRCH$BUF$SIZE];
        extern word       WRCT$BUF$EOB;
        extern word       WRCH$BUF$OUT;

#ifdef TESTING
printf("WRCH$BUF$OUT:    %d,    WRCT$BUF$EOB:    %d\n",    WRCH$BUF$OUT,
WRCT$BUF$EOB);
#endif
        while( WRCH$BUF$OUT != WRCT$BUF$EOB ) {

              printf("%c", WRCH$BUF$RING[WRCH$BUF$OUT]);
              /* wrap around index */
              WRCH$BUF$OUT += 1;
              if( WRCH$BUF$OUT == WRCH$BUF$SIZE )
                    WRCH$BUF$OUT= 0;
        }
        printf("\n");
} /* end Display$Buf */

/*
end */
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   isdn$constructor                         -
 -    Input       :   none                                     -
 -    Output      :   none                                     -
 -    Action      :   creates the ISDN layer task              -
 -    Date        :   21 Mar 90                                -
 -    UpDate      :   01 May 90                                -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
     ISDN interface to access basic rate digital communication.
*/
/*   Function declarations */
extern void        WISDN$TASK();

/*   interface mailbox to ISDN for requests */
token        WISDN$COMMAND$DMB;
token        WISDN$RESPONSE$DMB;
token        WISDN$STATUS$DMB;

void isdn$constructor()
{
       extern token       WISDN$COMMAND$DMB;
       extern token       WISDN$RESPONSE$DMB;
       extern token       WISDN$STATUS$DMB;
       word               status;

       /* create interface to ISDN layer */
       WISDN$COMMAND$DMB = rq$create$mailbox(DATA$TYPE, &status);
       WISDN$RESPONSE$DMB = rq$create$mailbox(DATA$TYPE, &status);
       WISDN$STATUS$DMB = rq$create$mailbox(DATA$TYPE, &status);

       /* create ISDN layer of the workstation operating system */
       rq$create$task(LOW$PRIORITY, WISDN$TASK, DATA$SEG, STACK$PTR,
                             STACK$SIZE, NO$FLOATS, &status);

} /*end isdn$constructor()*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   WISDN$TASK                               -
 -    Input       :   none                                     -
 -    Output      :   none                                     -
 -    Action      :   Provides access to basic rate interface. -
 -    Date        :   22 Mar 90                                -
 -    UpDate      :   01 May 90                                -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
word             fnc;
byte             mic_mode='\0';
byte             ear_mode='\0';
word             ear_vol=5;

void WISDN$TASK()
{
       word               num$bytes$recv;
       word               status;
       extern token       WISDN$COMMAND$DMB;

       while( FOREVER ) {
              num$bytes$recv =
                     rq$receive$data(WISDN$COMMAND$DMB, &fnc, FOREVER,
                     &status);
```

```
            if( num$bytes$recv != 2 )
                 printf("ISDN: Mailbox received -> %d \n",
                 num$bytes$recv);
#ifdef TRACING
            printf("FUNCTION: %u \n", fnc);
#endif

            switch( fnc ) {
                 case 0 : /* z */
                        Send_Command('z',NIL,NIL);
               break;
                 case 6 :
                        call_isdn_status();
                        break; /* v */
                 case 7 :
                        get_isdn_status();
                        break; /* v */
                 case 11 :
                        crn_voice = call_isdn_dial();
                        break; /* a */
                 case 12 :
                        call_isdn_voice_accept();
                        break; /* not implemented */
                 case 13 :
                        crn_voice = call_isdn_dial_disconnect();
                        break; /* b */
                 case 14 :
                        crn_voice = call_isdn_voice_reject();
                        break; /* c */
                 case 21 :
                        crn_data = call_isdn_connect( RATE_64K );
                        break; /* d */
                 case 22 :
                        crn_data = call_isdn_connect_accept();
                        break; /* not implemented */
                 case 23 :
                        crn_data = call_isdn_disconnect( crn_data );
                        breal-; /* e */
                 ca e 24 :
                        crn_data = call_isdn_connect_reject( crn_data );
                        break; /* f */
                 case 25 :
                        call_isdn_transmit( crn_data );
                        break; /* g */
                 case 26 :
                        call_isdn_receive_wait( crn_data );
                        break; /* h */
                 case 31 :
                        crn_second = call_isdn_connect( RATE_16K );
                        break; /* i */
                 case 32 :
                        crn_second = call_isdn_connect_accept();
                        break; /* not implemented */
                 case 33 :
                        crn_second = call_isdn_disconnect( crn_second );
                        break; /* j */
                 case 34 :
                        crn_second = call_isdn_connect_reject(crn_second);
                        break; /* k */
                 case 35 :
/*                      call_key_talk( crn_second ); */
                        break; /* l */
                        case 36 :
/*                              call_key_talk( crn_second ); */
                                break; /* m */
                        case 41 :
                                crn_file = call_isdn_connect( FILE_64K );
                                break; /* n */
                        case 42 :
                                crn_file = call_isdn_connect_accept();
                                break; /* not implemented */
                        case 43 :
                                crn_file = call_isdn_disconnect( crn_file );
                                break; /* o */
                        case 44 :
                                crn_file = call_isdn_connect_reject( crn_file );
                                break; /* p */
                        case 45 :
                                call_file_transmit();
                                break; /* q */
                        case 46 :
                                call_file_receive_wait();
                                break; /* r */
                        case 48 :
                                auto$file = 1;
                                call_file_transmit();
                                auto$file = 0;
                                break; /* q */
                        case 49 :
                                auto$file = 2;
                                call_file_transmit();
                                auto$file = 0;
                                break; /* q */
                        case 51:
                                call_isdn_mute_mic( mic_mode=((mic_mode+1)%2) );
                                break; /* s */
                        case 52:
                                call_isdn_mute_ear( ear_mode=((ear_mode+1)%2) );
                                break; /* t */
                        case 53:
                                call_isdn_audio_vol( ear_vol=((ear_vol+1)%10) );
                                break; /* u */
                        case 54:
                                call_isdn_audio_vol( ear_vol=((ear_vol-1)%10) );
                                break; /* u */

                        default:
                                printf("ISDN: unknown token -> %u \n", fnc);
                } /*end switch*/

        } /*end while*/

} /*end WISDN$TASK()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -   Function      :   call_isdn_dial                                    -
 -   Input         :   none                                              -
 -   Output        :   call reference number                             -
 -   Action        :   Initiates an ISDN voice call                      -
 -   Date          :   05 Apr 90                                         -
 -   UpDate        :                                                     -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Make an outgoing voice connection.                    */
```

```
int call_isdn_dial()
{
      word               status;

#ifdef TRACING
      printf("\n Making an ISDN Voice call. \n");
#endif

      /* command byte is 'a' to make a voice call */
      status = Send_Command('a',NIL,NIL);

      if( status < 0 )
            printf("Error - Make Voice Call : %d\n", status);
      return( status );

} /*end call_isdn_dial()*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function   :  call_voice_accept                                   -
 -    Input      :  none                                                -
 -    Output     :  none                                                -
 -    Action     :  Accepts an incomming ISDN voice call                -
 -    Date       :  05 Apr 90                                           -
 -    UpDate     :                                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* Accept an incoming voice connection is not implemented on the TE
   since the NT cannot initiate a call. */
void call_isdn_voice_accept()
{
      printf("\n Accepting an ISDN Voice call.*** \n");
} /*end call_isdn_voice_accept*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function   :  call_isdn_dial_disconnect                           -
 -    Input      :  none                                                -
 -    Output     :  clears call reference number                        -
 -    Action     :  Disconnects an ISDN voice call                      -
 -    Date       :  05 Apr 90                                           -
 -    UpDate     :                                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* Disconnect a voice call.                          */
int call_isdn_dial_disconnect()
{
      word               status;

#ifdef TRACING
      printf("\n Disconnecting an ISDN Voice call. \n");
#endif
     /* command byte is 'b' to disconnect a voice call */
      status = Send_Command('b',NIL,NIL);

      if( status < 0 )
            printf("Error - Disconnect Voice Call : %d\n", status);
      return( 0 );
} /*end call_isdn_dial_disconnect*/
```

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_voice_reject                         -
 -    Input       :   none                                           -
 -    Output      :   clears call reference number                   -
 -    Action      :   Rejects an ISDN voice call                     -
 -    Date        :   05 Apr 90                                      -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* Reject an incoming voice connection.              */
int call_isdn_voice_reject()
{
      word                status;

#ifdef TRACING
      printf("\n Rejecting an ISDN Voice call. \n");
#endif
     /* command byte is 'c' to reject a voice call */
      status = Send_Command('c',NIL,NIL);

      if( status < 0 )
           printf("Error - Reject Voice Call : %d\n", status);
      return( 0 );
} /*end call_isdn_voice_reject*/


/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_connect                              -
 -    Input       :   rate adaption                                  -
 -    Output      :   call reference number                          -
 -    Action      :   Initiates an ISDN data call                    -
 -    Date        :   05 Apr 90                                      -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* Establish a data connection on the B-channel.   */
int call_isdn_connect( rate )
      int                       rate;
{
      word                status;

#ifdef TRACING
      printf("\n Initiating an ISDN Data call. \n");
#endif
      if( rate == RATE_64K )
           /* command byte is 'd' to make a 64K data call */
           status = Send_Command('d',NIL,NIL);

      else if( rate == RATE_16K )
           /* command byte is 'i' to make a 16K second data call */
           status = Send_Command('i',NIL,NIL);

      else if( rate == FILE_64K )
      /* command byte is 'n' to make a 64K file transfer data call */
           status = Send_Command('n',NIL,NIL);

      else
           printf("Unknown rate specified\n");

      if( status < 0 )
           printf("Error - Make Data Call  : %d\n", status);
      return( status );
} /*end call_isdn_connect*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_connect_accept                  -
 -    Input       :   none                                      -
 -    Output      :   call reference number                     -
 -    Action      :   Accepts an ISDN data call                 -
 -    Date        :   05 Apr 90                                 -
 -    UpDate      :                                             -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Accept an incoming data connection is not implemented on the TE
   since the NT cannot initiate a call. */
int call_isdn_connect_accept()
{
     printf("\n Accepting an ISDN Data call.*** \n");
} /*end call_isdn_connect_accept*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_connect_reject                  -
 -    Input       :   CRN                                       -
 -    Output      :   call reference number                     -
 -    Action      :   Rejects an ISDN data call                 -
 -    Date        :   05 Apr 90                                 -
 -    UpDate      :                                             -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Reject an incoming data connection.              */
int call_isdn_connect_reject( id )
     int        id;
{
    word    status;

#ifdef TRACING
     printf("\n Rejecting an ISDN Data call. \n");
#endif
     if( id == crn_data )
          /* command byte is 'f' to reject a 64K data call */
          status = Send_Command('f',NIL,NIL);

     else if( id == crn_second )
          /* command byte is 'k' to reject a 16K second data call */
          status = Send_Command('k',NIL,NIL);

     else if( id == crn_file )
          /* command byte is 'p' to reject a file data call */
          status = Send_Command('p',NIL,NIL);

     else
          printf("Unknown CRN specified\n");

     if( status < 0 )
          printf("Error - Data Call Reject  : %d\n", status);
     return(0);
} /*end call_isdn_connect_reject*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_disconnect                      -
 -    Input       :   call refernce number                      -
 -    Output      :   clear call reference number               -
 -    Action      :   Disconnects an ISDN data call             -
 -    Date        :   05 Apr 90                                 -
 -    UpDate      :                                             -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

```
/* Disconnects a data connection.  */
int call_isdn_disconnect( id )
int id;
{
      word  status;
      word  rep_stat;

#ifdef TRACING
      printf("\n Disconnecting an ISDN Data call. \n");
#endif

      if( id == crn_data ) {
            /* command byte is 'e' to reject a 64K data call */
            rep_stat = Send_Command('e',NIL,NIL);
            rq$send$data(WISDN$STATUS$DMB, &rep_stat, 2, &status);
      }

      else if( id == crn_second )
            /* command byte is 'j' to reject a 16K second data call */
            status = Send_Command('j',NIL,NIL);

      else if( id == crn_file )
            /* command byte is 'o' to reject a file data call */
            status - Send_Command('o',NIL,NIL);

      else
            printf("Unknown Call Ref. No. specified\n");

      if( status < 0 )
            printf("Error - Disconnect Data Call : %d\n", status);
      return(0);
} /*end call_isdn_disconnect*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   send_isdn_transmit                              -
 -    Input       :   none                                            -
 -    Output      :   none                                            -
 -    Action      :   Sends a string of characters                    -
 -    Date        :   05 Apr 90                                       -
 -    UpDate      :   01 May 90                                       -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Transmits the buffer of characters entered from the user interface. */
void send_isdn_transmit()
{
      word  bfr_lgth;

      printf("Please enter data\n");
      gets(bfr);

      if  ( strlen(&bfr[0]+1) < BURST$SIZE )
            bfr_lgth = strlen(&bfr[0]) + 1;
      else
            bfr_lgth = BURST$SIZE;
#ifdef TRACING
      printf("Length sent : %u\n",bfr_lgth);
      printf("Bfr sent : %s\n", bfr);
#endif

      /* send length of string */
      Send$Word( bfr_lgth );
      /* send actual data string */
      Send$Buf( bfr );
} /*end send_isdn_transmit*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :    call_isdn_transmit                           -
 -    Input        :    call reference number                        -
 -    Output       :    none                                         -
 -    Action       :    Sends the command request for a data transfer -
 -    Date         :    05 Apr 90                                    -
 -    UpDate       :                                                 -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Sends the command request for a data string transfer. */
void call_isdn_transmit( id )
      word  id;
{
      word  status;

#ifdef TRACING
      printf("\n Transmitting command in ISDN Data call. \n");
#endif
      if( id == crn_data )
            /* command byte is 'g' to transmit data on a 64K data call */
            status = Send_Command('g',NIL,NIL);

      else if( id == crn_second )
      /* command byte is 'l' to transmit data on a 16K second data call */
            status = Send_Command('l',NIL,NIL);

      else
            printf("Unknown Call Ref. No. specified\n");

      if( status < 0 )
            printf("Error - Transmit Data : %d\n", status);
} /*end call_isdn_transmit*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :    call_file_transmit                           -
 -    Input        :    call reference number                        -
 -    Output       :    none                                         -
 -    Action       :    Sends the command request for a file transfer -
 -    Date         :    05 Apr 90                                    -
 -    UpDate       :    12 Apr 90                                    -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Sends the command request for a data file transfer. */
void call_file_transmit()
{
      word  status;

#ifdef TRACING
      printf("\n Transmitting file command in ISDN Data call. \n");
#endif
      /* command byte is 'q' to transmit a file on a 64K data call */
      status = Send_Command('q',NIL,NIL);

      if( status < 0 )
            printf("Error - Transmit FILE TRANSFER : %d\n", status);
} /*end call_file_transmit*/
```

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   send_file_transmit                             -
 -    Input       :   none                                           -
 -    Output      :   none                                           -
 -    Action      :   Sends the data file for a file transfer        -
 -    Date        :   05 Apr 90                                      -
 -    UpDate      :   01 May 90                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* Transmits the buffer of characters specified by the user's filename. */
void send_file_transmit()
{
      word  numread;
      FILE  *fp;                          /* file pointer to open file */
      byte  filename[81];                 /* name of file to open */
      byte  done;

      if( !auto$file ) {
            scanf("%c",&done); /* clear buffer */
            printf("Please enter the name of the file to transmit: ");
            gets(filename);
      }
      else if( auto$file == 1 )
            strcpy(filename,"helpfile.dat");
      else
            strcpy(filename,"whofile.dat");

      while( (fp=fopen(filename, "r")) == NULL) {
                  printf("Error opening file: %s\n",filename);
            printf("Please enter the name of the file to transmit: ");
            gets(filename);
      } /*end while*/

      done = FALSE;
      while( !done) {
            numread = fread( (char*)bfr,sizeof(char),BURST$SIZE,fp);
            /* transmit buffer length */

#ifdef TESTING
      wait();
      printf("numread: %d\n",numread);
#endif

            Send$Word(numread);

            /* transmit buffer */
            Send$Num$Bytes(bfr, numread);

#ifdef TESTING
      wait();
      printf("ISDN: yes buffer has been sent\n");
#endif

            if( numread < BURST$SIZE )
                  if( feof(fp) != 0 )
                        done = TRUE;
      } /*end while*/

#ifdef TESTING
      wait();
      printf("about to close file\n"):
#endif
```

```
      /* end of file when numread < BURST$SIZE */
      fclose(fp);
} /*end send_file_transmit*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_receive_wait                      -
 -    Input       :   call refernece number                       -
 -    Output      :   none                                        -
 -    Action      :   Sends the command request to receive data   -
 -    Date        :   05 Apr 90                                   -
 -    UpDate      :   06 Apr 90                                   -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Receives the buffer of characters sent by the user. */
void call_isdn_receive_wait( id )
      word  id;
{
      word  status;

#ifdef TRACING
      printf("\n Transmitting receive command in ISDN Data call. \n");
#endif
      if( id == crn_data )
            /* command byte is 'h' to receive data on a 64K data call */
            status = Send_Command('h',NIL,NIL);

      else if( id == crn_second )
      /* command byte is 'm' to receive data on a 16K second data call */
            status = Send_Command('m',NIL,NIL);

      else
            printf("Unknown Call Ref. No. specified\n");

      if( status < 0 )
            printf("Error - Receive Data : %d\n", status);
} /*end call_isdn_receive_wait*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_file_receive_wait                      -
 -    Input       :   call reference number                       -
 -    Output      :   none                                        -
 -    Action      :   Sends the command request to receive a file -
 -    Date        :   05 Apr 90                                   -
 -    UpDate      :                                               -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Send the request command to receive a data file. */
void call_file_receive_wait()
{
      word  status;

#ifdef TRACING
      printf("\n Sending the file receive command in ISDN Data call. \n");
#endif
      /* command byte is 'r' to receive a file on a 64K data call */
      status = Send_Command('r',NIL,NIL);
      if( status < 0 )
            printf("Error - Receive FILE TRANSFER : %d\n", status);
} /*end call_file_receive_wait*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -     Function    :   send_file_receive                            -
 -     Input       :   none                                         -
 -     Output      :   none                                         -
 -     Action      :   Receives the data during a file transfer     -
 -     Date        :   05 Apr 90                                    -
 -     UpDate      :   12 Apr 90                                    -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/* Receives the file sent by the user. */
char  header[HEAD$SIZE];
void send_file_receive()
{
      extern token       WISDN$RESPONSE$DMB;
      extern byte        WRCH$BUF$RING[WRCH$BUF$SIZE];
      extern word        WRCH$BUF$OUT;
      FILE               *fp;              /* file pointer to open file */
      byte               filename[81];     /* name of file to open */
      word               bytecount,i;
      word               status;

      wait();
      printf("Please enter the name of the file to receive: ");
      gets(filename);
      printf("/n");

      while( (fp=fopen(filename, "w")) == NULL) {
            printf("Error opening file: %s\n",filename);
            printf("Please enter the name of the file to receive: ");
            gets(filename);
            printf("/n");
      } /*end while*/

      /* receive one file buffer request */
      rq$receive$data(WISDN$RESPONSE$DMB, header, FOREVER, &status);

      while( (bytecount = * (int *) &header[1]) == BURST$SIZE ) {
            fwrite( &WRCH$BUF$RING[WRCH$BUF$OUT], sizeof(char), bytecount,
                        fp);
            for(i=0; i<bytecount; i++) {
                  /* wrap around index */
                  WRCH$BUF$OUT +=1;
                  if( WRCH$BUF$OUT == WRCH$BUF$SIZE )
                        WRCH$BUF$OUT = 0;
        }
            /* receive one file buffer request */
            rq$receive$data(WISDN$RESPONSE$DMB, header, FOREVER, &status);
      } /*end while*/

      fwrite( &WRCH$BUF$RING[WRCH$BUF$OUT], sizeof(char), bytecount, fp);
      for(i=0; i<bytecount; i++) {
            /* wrap around index */
            WRCH$BUF$OUT +=1;
            if( WRCH$BUF$OUT == WRCH$BUF$SIZE )
                  WRCH$BUF$OUT = 0;
      }
      fclose(fp);
      printf("You have just received the following file:\n");
      display_file(filename);
} /*end send_file_receive*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   display_file                                -
 -    Input        :   filename                                    -
 -    Output       :   none                                        -
 -    Action       :   writes the data file to the standard output -
 -    Date         :   12 Apr 90                                   -
 -    UpDate       :          90                                   -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Writes the data file specified by the user to the standard output
device */
void display_file( filename )
      byte  filename[81];                /* name of file to open */
{
      word  numread,i;
      FILE  *fp;                         /* file pointer to open file */


      if( (fp=fopen(filename, "r")) == NULL) {
            printf("Error opening file: %s\n",filename);
            printf("Please enter the name of the file to display: ");
            gets(filename);
      }

      while( feof(fp) == FALSE ) {
            numread = fread( (char*)bfr,sizeof(char),BURST$SIZE,fp);
            for(i=0; i<numread; i++)
                  printf("%c",bfr[i]);
      } /*end while*/
      printf("/n");
      fclose(fp);
} /*end display_file*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function     :   call_isdn_audio_volume                      -
 -    Input        :   volume                                      -
 -    Output       :   none                                        -
 -    Action       :   Adjusts earpiece volume gain                -
 -    Date         :   05 Apr 90                                   -
 -    UpDate       :                                               -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Adjust earpiece volume gain.                    */
void call_isdn_audio_vol( volume )
      word  volume;
{
      word  status;

#ifdef TRACING
      printf("\n Adjusting the Earpiece volume. \n");
#endif
      status = Send_Command('u',NIL,volume);
      if( status < 0 )
            printf("Error - Adjust Volume : %d\n", status);
} /*end call_isdn_audio_vol*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_mute_mic                             -
 -    Input       :   mode                                           -
 -    Output      :   none                                           -
 -    Action      :   Enables/Disables handset microphone            -
 -    Date        :   05 Apr 90                                      -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Enable/Disable the handset microphone. 0=ON, 1=OFF      */
void call_isdn_mute_mic( mode )
      byte  mode;
{
      word  status;

#ifdef TRACING
      printf("\n Controlling the handset microphone. \n");
#endif
      status = Send_Command('s',mode,NIL);
      if( status < 0 )
            printf("Error - Mute Microphone : %d\n", status);
} /*end call_isdn_mute_mic*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   call_isdn_mute_ear                             -
 -    Input       :   mode                                           -
 -    Output      :   none                                           -
 -    Action      :   Enables/Disables handset earpiece              -
 -    Date        :   05 Apr 90                                      -
 -    UpDate      :                                                  -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Enable/Disable the handset earpiece. 0=ON, 1=OFF        */
void call_isdn_mute_ear( mode )
      byte  mode;
{
      word  status;

#ifdef TRACING
      printf("\n Controlling the handset earpiece. \n");
#endif
      status = Send_Command('t',mode,NIL);
      if( status < 0 )
            printf("Error - Mute Earpiece : %d\n", status);
} /*end call_isdn_mute_ear*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 -    Function    :   Send_Command                                   -
 -    Input       :   ISDN command, param BYTE, param WORD           -
 -    Output      :   status                                         -
 -    Action      :   Sends ISDN commands throuoh the PDLC           -
 -    Date        :   05 Apr 90                                      -
 -    UpDate      :   06 Apr 90                                      -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Sends ISDN commands through the PDLC.  */
int Send_Command( cmd, data_val, num_val)
      byte  cmd;
      byte  data_val;
      word  num_val;
{
      extern token      WISDN$RESPONSE$DMB;
      extern word       WRCT$BUF$EOB;
```

```
        extern word        WRCH$BUF$OUT;
        word               request$token;
        word               status;

        /* send the command */
        Send$Byte( cmd );

        /* receive parameters for certain function call commands */
        if( (cmd == 's') || (cmd == 't') )
              /* send the parameter data BYTE */
              Send$Byte( data_val );

        else if( (cmd == 'u') || (cmd == 'v') )
              /* send the parameter data WORD */
              Send$Word( num_val );

        switch( cmd ) {
              case 'g': case 'l':     send_isdn_transmit();
                                            break;
              case 'q':               send_file_transmit();
                                            break;
              case 'r':               send_file_receive();
                                            break;
              case 'z':               exit(0);
                                            break;
        } /*end switch*/

        /* receive status information */
        rq$receive$data(WISDN$RESPONSE$DMB, header, FOREVER, &status);
        request$token = * (int *) &header[1];

        if( header[0] == STAT$REQ )
                    return(request$token);
        else if( header[0] == BUF$REQ ) {
                    printf("Out of synchronization.\n");
                    Display$Buf();
                    /* receive status information */
                    rq$receive$data(WISDN$RESPONSE$DMB,  header,  FOREVER,
                                         &status);
                    request$token = * (int *) &header[1];
                    return(request$token);
        }
        else
                    printf("ISDN: unknown cmd returned %c\n", header[1]);

} /*end Send_Command*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_isdn_status                            -
-    Input           :   none                                        -
-    Output          :   none                                        -
-    Date            :   31 Oct 89                                   -
-    UpDate          :   01 May 90                                   -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Display the status of all connections using local information. */
void call_isdn_status()
{
int          status;

     if( crn_voice == 0 )
         printf("Voice connection is INACTIVE.\n");
        else {
        /* command byte is 'v' to obtain status information */
            status = Send_Command('v',NIL,crn_voice);
            if( status > 0 )
                printf("Voice connection is ACTIVE.\n");
            else {
                printf("Voice connection is INACTIVE.\n");
            /* update crn information */
                crn_voice = 0;
            }
        }

        if( crn_data == 0 )
            printf("Data (64 Kbps) connection is INACTIVE.\n");
        else {
            status = Send_Command('v',NIL,crn_data);
            if( status > 0 )
                printf("Data (64 Kbps) connection is ACTIVE.\n");
            else {
                printf("Data (64 Kbps) connection is INACTIVE.\n");
            /* update crn information */
                crn_data = 0;
            }
        }

        if( crn_second == 0 )
            printf("Second data (16 Kbps) connection is INACTIVE.\n");
        else {
            status = Send_Command('v',NIL,crn_second);
            if( status > 0 )
                printf("Second data (16 Kbps) connection is ACTIVE.\n");
            else {
                printf("Second data (16 Kbps) connection is INACTIVE.\n");
            /* update crn information */
                crn_second = 0;
            }
        }

        if( crn_file == 0 )
            printf("File transfer data connection is INACTIVE.\n");
        else {
            status = Send_Command('v',NIL,crn_file);
            if( status > 0 )
                printf("File transfer data connection is ACTIVE.\n");
            else {
                printf("File transfer data connection is INACTIVE.\n");
            /* update crn information */
                crn_file = 0;
            }
        }

} /*end call_isdn_status*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-     Function        :   get_isdn_status                           -
-     Input           :   none                                      -
-     Output          :   none                                      -
-     Date            :   12 Apr 90                                 -
-     UpDate          :   16 Apr 90                                 -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Display the status of all connections using local information. */
```

```
void get_isdn_status()
{
int          status;

      if( crn_voice != 0 ) {
     /* command byte is 'v' to obtain status information */
         status = Send_Command('v',NIL,crn_voice);
         if( status <= 0 )
         /* update crn information */
             crn_voice = 0;
      }

      if( crn_data != 0 ) {
         status = Send_Command('v',NIL,crn_data);
         if( status <= 0 )
         /* update crn information */
             crn_data = 0;
      }

      if( crn_second != 0 ) {
         status = Send_Command('v',NIL,crn_second);
         if( status <= 0 )
         /* update crn information */
             crn_second = 0;
      }

      if( crn_file != 0 ) {
         status = Send_Command('v',NIL,crn_file);
         if( status <= 0 )
         /* update crn information */
             crn_file = 0;
      }

} /*end get_isdn_status*/

/*
end file isdn.c */
```

```
      name  mouse
;[]-------------------------------------------------------------------[]
;|  Project        :   ISDN Personal Workstation                        |
;|  Sub-Project    :   Hardware Interrupt and PPI Support               |
;|  File           :   mouseint.asm                                     |
;|  Author         :   Zenon Slodki                                     |
;|  Date           :   13 Feb 1990                                      |
;|  Update         :   23 Mar 1990                                      |
;|                                                                      |
;[]-------------------------------------------------------------------[]
;
; declaration of external RMX Nucleus functions */
extrn rqsignalinterrupt:far
;
ASSUME DS:support_DATA, SS:STACK
STACK STACKSEG 400
LEVELA            EQU 24H                ;IRQ12
LEVELB            EQU 23H                ;IRQ11
LEVELS            EQU 27H                ;IRQ15
PORTCT            EQU 3A3H               ;Port C I/O address
RSTINTB           EQU 0CH                ;00001100
RSTINTA           EQU 0EH                ;00001110

support_DATA SEGMENT rw public
errora            DW    0
errorb            DW    0
errors            DW    0
support_DATA ENDS
;
_TEXT SEGMENT eo public
ASSUME  ds:support_DATA, ss:STACK

; This is the INTERRUPT HANDLER for any spurious interrupts on the slave
PIC.
; The function of this handler is to signal its supporting interrupt task.
PUBLIC spurhand
spurhand PROC FAR
      push  bp
      mov   bp,sp
      push  ax
      push  ds
      mov   ax, support_DATA  ; put the address of the data
      mov   ds, ax            ; segment into data segment reg.
      mov   ax,LEVELS         ; first parameter is the interrupt
      push  ax                ; level value
      lea   ax,errors
      push  ds                ; second parameter is the error
      push  ax                ; status return pointer
      call  rqsignalinterrupt ; notify interrupt service task

      pop   ds
      pop   ax
      pop   bp
      iret
spurhand ENDP

; This is the INTERRUPT HANDLER for the ISDN MOUSE Int A.
; The function of this handler is to signal its supporting interrupt task.
PUBLIC intahand
intahand PROC FAR
      push  bp
      mov   bp,sp
      push  ax
      push  dx
      push  ds
      mov   ax, support_DATA  ; put the address of the data
      mov   ds, ax            ; segment into data segment reg.

; disable further INT A interrupts by reseting the interrupt hold
; flip-flop
      mov   al,RSTINTA
      mov   dx,PORTCT
      out   dx,al

      mov   ax,LEVELA         ; first parameter is the interrupt
      push  ax                ; level value
      lea   ax,errora
      push    ds              ; second parameter is the error
      push    ax              ; status return pointer
      call    rqsignalinterrupt     ; notify interrupt service task
      pop     ds
      pop     dx
      pop     ax
      pop     bp
      iret
intahand ENDP

; This is the INTERRUPT HANDLER for the ISDN MOUSE Int B.
; The function of this handler is to signal its supporting interrupt task.
PUBLIC intbhand
intbhand PROC FAR
      push  bp
      mov   bp,sp
      push  ax
      push  dx
      push  ds
      mov   ax, support_DATA  ; put the address of the data
      mov   ds, ax            ; segment into data segment reg.

; disable further INT B interrupts by reseting the interrupt hold
; flip-flop
      mov   al,RSTINTB
      mov   dx,PORTCT
      out   dx,al

      mov   ax,LEVELB         ; first parameter is the interrupt
      push  ax                ; level value
      lea   ax,errorb
      push  ds                ; second parameter is the error
      push  ax                ; status return pointer
      call  rqsignalinterrupt ; notify interrupt service task

      pop   ds
      pop   dx
      pop   ax
      pop   bp
      iret
intbhand ENDP

; This procedure sends a byte of data to the specified port address. Both
; parameters are passed on the stack.
PUBLIC output
output PROC FAR
      push  bp
```

```
        mov   bp,sp
        push  ax
        push  dx
        mov   dx,[bp+6]
        mov   al,[bp+8]
        out   dx,al
        pop   dx
        pop   ax
        pop   bp
        ret
output ENDP

_TEXT ENDS

END
```

```
      name  pdlcint
;
;[]-----------------------------------------------------------------[]
;|  Project       : ISDN Personal Workstation                          |
;|  Sub-Project   : Parallel Communication Protocol                    |
;|  File          : pdlcint.asm                                        |
;|  Author        : Zenon Slodki                                       |
;|  Date          : 29 Mar 1990                                        |
;|  Update        : 11 Apr 1990                                        |
;|                                                                     |
;[]-----------------------------------------------------------------[]
;
; declaration of external RMX Nucleus functions */
extrn rqsignalinterrupt:far
extrn rqexitinterrupt:far
extrn error_hand:far
;
ASSUME DS:pdlcint_DATA, SS:STACK
STACK STACKSEG 400
WRCHBUFSIZE EQU 9000
WTRHBUFSIZE EQU 9000
PORTA       EQU 03A0H
PORTB       EQU 03A1H
RECVLEVEL   EQU 58H
TRANLEVEL   EQU 22H
HEADSIZE    EQU 3
READYMODE   EQU 0
;
pdlcint_DATA SEGMENT rw public
;
; Receiver buffer declarations
PUBLIC WRCHBUFIN
WRCHBUFIN   DW    0           ; input pointer to receiver buffer
PUBLIC WRCHBUFOUT
WRCHBUFOUT  DW    0           ; output pointer to receiver buffer
PUBLIC WRCHBUFRING
WRCHBUFRING DB    WRCHBUFSIZE DUP(?) ; the receiver ring buffer
PUBLIC WRCTBUFEOB
WRCTBUFEOB  DW    0           ; end of buffer pointer for recv.
;
PUBLIC WRCHHEADGLB
WRCHHEADGLB DB    3 DUP(?)    ; receiver header
PUBLIC WRCTMODEGLB
WRCTMODEGLB DW    0           ; receiver modes
PUBLIC WRCTCOUNTGLB
WRCTCOUNTGLB DW   0           ; byte counter for header
;
; Transmitter buffer declarations
PUBLIC WTRHBUFIN
WTRHBUFIN   DW    0           ; input pointer to transmitter buffer
PUBLIC WTRHBUFOUT
WTRHBUFOUT  DW    0           ; output pointer to transmitter buffer
PUBLIC WTRHBUFRING
WTRHBUFRING DB    WTRHBUFSIZE DUP(?) ; the transmitter ring buffer
PUBLIC WTRTBUFEOB
WTRTBUFEOB  DW    0           ; end of buffer pointer for trans.
;
;
error       DW  0
pdlcint_DATA ENDS
;
_TEXT SEGMENT eo public
```

```
ASSUME      ds:pdlcint_DATA, ss:STACK

; This is the RECEIVER INTERRUPT HANDLER which reads the data byte from
; PORT A then places the data into a ring buffer. Next it signals the
; interrupt service task to complete the processing.
PUBLIC WRCHTASK
WRCHTASK PROC FAR
      push  bp
      mov   bp,sp
      push  ax
      push  dx
      push  si
      push  ds
      mov         ax, pdlcint_DATA          ; put the address of the data
      mov         ds, ax                    ; segment into data segment reg.
      mov         dx,PORTA                  ; get port address
      in          al,dx                     ; read data byte
      cmp         WRCTMODEGLB,READYMODE     ; determine how to interpret
                                            ; data
      jne         BUFMODE

; header processing
      mov         si,WRCTCOUNTGLB           ; get header pointer
      mov         WRCHHEADGLB[si],al        ; store byte in header
      inc         si                        ; adjust pointer
      mov         WRCTCOUNTGLB,si           ; update header pointer
      cmp         si,HEADSIZE               ; received complete header?
      je          done
      jmp         moreinfo

; buffer processing
BUFMODE:
      mov         si,WRCHBUFIN              ; get buffer pointer
      mov         WRCHBUFRING[si],al        ; store byte in receiver buffer
      inc         si                        ; adjust pointer
      cmp         si,WRCHBUFSIZE            ; wrap around buffer pointer
      jne         not_wrap                  ; not wrapping so jump
      xor         si,si                     ; yes, reset buffer pointer
not_wrap:
      mov         WRCHBUFIN,si              ; save updated buffer pointer
      cmp         si,WRCTBUFEOF             ; received complete buffer?
      je          done

; the handler must send an end of interrupt signal to the PIC
moreinfo:
      mov         ax,RECVLEVEL              ; first parameter is the
                                            ; interrupt
      push  ax                              ; level value
      lea         ax,error
      push  ds                              ; second parameter is the error
      push  ax                              ; status return pointer
      call  rqexitinterrupt                 ; send an end of interrupt
                                            ; signal
      jmp         fin

done:
      mov         ax,RECVLEVEL              ;  first   parameter   is   the
interrupt
      push  ax                              ; level value
      lea         ax,error
      push  ds                              ; second parameter is the error
      push  ax                              ; status return pointer
      call   rqsignalinterrupt              ; notify interrupt service task

fin:
      pop       ds
      pop       si
      pop       dx
      pop       ax
      pop       bp
      iret
WRCHTASK ENDP


; This is the TRANSMITTER INTERRUPT HANDLER which sends the data byte from
; PORT B. It sends a byte at a time until the buffer becomes empty, then
; it notifies the interrupt task that the buffer has been completely sent.
PUBLIC WTRHTASK
WTRHTASK PROC FAR
      push  bp
      mov   bp,sp
      push  ax
      push  dx
      push  si
      push  ds
      mov           ax, pdlcint_DATA       ; put the address of the data
      mov           ds, ax                 ; segment into data segment reg.
      mov           si,WTRHBUFOUT          ; get buffer pointer
      cmp           WTRHBUFIN,si           ; empty buffer error check
      je            intrans_error
;
      xor           ax,ax
      mov           al,WTRHBUFRING[si]     ; send this byte
      mov           dx,PORTB               ; get port address
      out           dx,al                  ; send data byte
;
      inc           si                     ; adjust buffer pointer
      cmp           si,WTRHBUFSIZE         ; wrap around buffer pointer
      jne           neg_wrap               ; not wrapping so jump
      xor           si,si                  ; yes, reset buffer pointer
neg_wrap:
      mov           WTRHBUFOUT,si          ; update buffer pointer
;
      cmp           si,WTRTBUFEOB          ; is this the last byte?
      je            last_byte              ; yes, so signal$interrupt

; the handler must send an end of interrupt signal to the PIC
      mov           ax,TRANLEVEL           ;  first  parameter  is  the
                                           ; interrupt
      push  ax                             ; level value
      lea           ax,error
      push  ds                             ; second parameter is the error
      push  ax                             ; status return pointer
      call  rqexitinterrupt                ;  send  an  end  of  interrupt
                                           ; signal
      jmp           fin2

last_byte:
      mov           ax,TRANLEVEL           ;  first  parameter  is  the
                                           ; interrupt
      push  ax                             ; level value
      lea           ax,error
      push  ds                             ; second parameter is the error
      push  ax                             ; status return pointer
      call  rqsignalinterrupt              ; notify interrupt service task

fin2:
      pop        ds
      pop        si
      pop        dx
      pop        ax
      pop        bp
      iret
WTRHTASK ENDP
;
;
intrans_error:
      call  ip_load                        ; place the ip on the stack
ip_load:
      push  cs
      mov        ax,1                      ; error handler condition code
      push  ax                             ; return this value
      call  error_hand                     ; goto a C routine


_TEXT ENDS

END
```

IBM PC ISDN Server Software Listing

```
PAGE, 132
        NAME cint
;[]-------------------------------------------------------------[]
;|      Author      : Zenon Slodki                              |
;|      Project     : Distributed ISDN communication            |
;|      File        : cint.asm                                  |
;|      Date        : 08 Dec 1989                               |
;|      Update      : 16 Apr 1990                               |
;|      Version     : 1                                         |
;|                                                              |
;[]-------------------------------------------------------------[]
;       Description:
;       This is a stand alone routine that sends data to the
;       PC's PPI using an interrupt handler and storing data in a
;       circular buffer.
;       These routines are based on 'talk' by Ray Duncan, Advanced MSDOS.
;
;
;       Macros
;
OSCall  MACRO   DosFunct        ; calls the Operating System
        push    ax
        mov     ah, DosFunct
        int     21h
        pop     ax
        ENDM

Write   MACRO   Char            ; displays one character
        push    dx
        mov     dl, Char        ; destroys DL
        OSCall  2
        pop     dx
        ENDM

Wr_str  MACRO   Str_pnt         ; uses DOS call to display string
        push    dx
        mov     dx, OFFSET Str_pnt ; DS:DX point to string
        OSCall  09h             ; Print String
        pop     dx
        ENDM

OPort   MACRO  Port,Val
        push    dx
        mov dx,Port
        mov al,Val
        out dx, al
        pop     dx
        ENDM

IPort MACRO Port
        push    dx
        mov dx, Port
        in  al,dx
        pop     dx
        ENDM
;
;       constants
;
LF       EQU     0ah
CR       EQU     0dh
```

```
End_st   EQU     '$'                ; DOS end of string
Pic_mask EQU     21h                ; port address, 8259 mask register
Pic_eoi  EQU     20h                ; port address, 8259 eoi instr. control
reg
Int_mask EQU     20h                ; Mask for 8259 IRQ5 (receiver)
Buf_len  EQU     9000               ;
com_int  EQU     0dh                ; interrupt no. for IRQ5

;
;       Port constants
;
Base_port        EQU     03A0h
Port_A           EQU     Base_port + 0
Port_B           EQU     Base_port + 1
Port_C           EQU     Base_port + 2
Port_Ct          EQU     Base_port + 3

Run_mode         EQU     10110100b  ; port A input, port B output, Mode 1
Int_A            EQU     00001001b  ; input interrupt enable for port A
Int_B            EQU     00000101b  ; output interrupt enable for port B
Data_out         EQU     00000001b  ; determine when output buffer is
                                    ; available
;                        76543210

DGROUP       GROUP _DATA
;
_TEXT SEGMENT byte public 'CODE'
      ASSUME cs:_TEXT,ds:DGROUP

;
;
; Procedure replaces the IRQ5 interrupt vector with a custom
; interrupt handler and enables interrupts from the 8259.
PUBLIC _Install
_Install PROC    FAR
        push    bp              ; preserve bp in order to use the bp
        mov     bp,sp           ; to access parameters on the stack
        push    ax
        push    dx
        push    ds
        mov     al,com_int      ; get interrupt handler address
        OSCall  35h             ; ES:BX = addres
        mov     intc_seg,es     ; save segment
        mov     intc_offs,bx    ; save offset
        mov     dx,offset int_han ; address of custom intrp. handler
        push    cs
        pop     ds
        mov     al,com_int      ; interrupt number

        OSCall  25h            ; set address of the new custom interrupt
                                ; handler routine for specified machine
                                ; intrp
        pop     ds
        in      al,Pic_mask     ; read current 8259 intrp. mask
        and     al,not Int_mask ; reset mask for this Com port
        out     Pic_mask,al     ; write back 8259 intrp. mask

; initialize PPI to mode 1 and enable interrupts
        OPort   Port_Ct, Run_mode
        OPort   Port_Ct, Int_A
        OPort   Port_Ct, Int_B
        pop     dx                ; restore registers and return
        pop     ax
        pop     bp
        ret
_Install ENDP

;
;
; Procedure restores the previous IRQ5 interrupt vector
; and enables interrupts from the 8259.
PUBLIC _UN_Install
_UN_Install PROC    FAR
        push    bp                ; preserve bp in order to use the bp
        mov     bp,sp             ; to access parameters on the stack
        push    ax
        push    dx
        in      al,Pic_mask       ; read current 8259 intrp. mask
        or      al,Int_mask       ; set mask for this Com port
        out     Pic_mask,al       ; write back 8259 intrp. mask
        push    ds
        mov     dx,intc_offs      ; saved offset
        mov     ds,intc_seg       ; saved segment
        mov     al,com_int        ; interrupt number
        OSCall  25h              ; restore original addr. of intrp. handler
        pop     ds
        pop     dx                ; restore registers and return
        pop     ax
        pop     bp
        ret
_UN_Install ENDP

;
;
; Procedure int_han is a custom interrupt handler for the serial port
interrupt
; which places received characters into a ring buffer.
int_han PROC    FAR
        sti                       ; turn interrupts back on
        push    ax                ; save all necessary registers
        push    bx                ;
        push    dx                ;
        push    ds                ;
        mov     ax, seg DGROUP   ; put the address of the data segment into
        mov     ds, ax            ; the data segment register
        mov     dx,Port_A         ;
        in      al,dx             ; read this character
        cli                       ; clear interrupts for pointer
                                  ; manipulation
        mov     bx,asc_in         ; get buffer pointer
        mov     [_Asc_buf+bx],al ; store this character
        inc     bx                ; adjust pointer
        cmp     bx,Buf_len        ; wrap around buffer pointer
        jne     no_wrap           ; no, jump
        xor     bx,bx             ; yes, reset buffer pointer
no_wrap:
        mov     asc_in,bx         ; save updated buffer pointer
        sti
        call    rmx_delay         ; Give rmx a chance to catch its breath

        mov     al,20h            ; send NEOI to 8259
        out     Pic_eoi al        ;
```

```
; restore all  registers
        pop     ds              ;
        pop     dx              ;
        pop     bx              ;
        pop     ax              ;
        iret                    ; exit interrupt handler
int_han ENDP

;
;
; Procedure Send_Buf send a string of characters through the port.
; The string must terminate in a 0 (ASCIIZ).
; This terminating 0 byte is sent.
PUBLIC  _Send_Buf
_Send_Buf PROC    FAR
        push    bp              ; preserve bp in order to use the bp
        mov     bp,sp           ; to access parameters on the stack
        push    si
        pushf
        push    ax
        push    cx
        push    dx
        push    ds
;    mov     ax, seg DGROUP    ; put the address of the data segment into
        mov     ax, [bp+8]     ; put the address of the data segment into
                                ; the data segment selector is after the
                                ; offset on the calling stack
        mov     ds, ax          ; the data segment register
        cld
        mov     si,[bp+6]       ; get starting address of the string
string_loop:
        lodsb                   ; get a character into the al register
        or      al,al           ; check for terminating 0 byte
        jz      end_string      ; reached the end of the string
        push    ax
inner_loop:
        IPort   Port_C
        and     al,Data_out
        jz      inner_loop
        call    out_delay       ; Used as a necessary delay
        pop     ax
        OPort   Port_B,al       ; transmit the character byte
        jmp     string_loop     ; continue with next character
end_string:
        IPort   Port_C
        and     al,Data_out
        jz      end_string
        call    out_delay       ; Used as a necessary delay
        xor     ax,ax
        OPort   Port_B,al       ; transmit the zero character byte
      pop ds
        pop     dx              ; restore registers and return
        pop     cx
        pop     ax
        popf
        pop     si
        pop     bp
        ret
_Send_Buf ENDP

;
;
```

```
; Procedure out_delay used as a wait state generator
out_delay PROC   NEAR
        push     ax              ; save all necessary registers
        mov      ax,04fh         ; 0fh good but 01 is too low
uphere: nop
        dec      ax
        jnz      uphere
        pop      ax              ; restore all registers
        ret                      ;
out_delay ENDP

;
;
; Procedure out_delay used as a wait state generator
rmx_delay PROC   NEAR
        push     ax              ; save all necessary registers
        push     bx
        mov      ax,01ffh
gomore: mov      bx,0fh
upmore: nop
        dec      bx
        jnz      upmore
        dec      ax
        jnz      gomore
        pop      bx              ; restore all registers
        pop      ax
        ret                      ;
rmx_delay ENDP

;
;
; Procedure Buf_stat test whether characters from the port are waiting
; in the ring buffer. Returns a 1 in the AX register if true else 0
; if nothing waiting.
PUBLIC _Buf_stat
_Buf_stat PROC  FAR
        push     bp              ; preserve bp in order to use the bp
        mov      bp,sp           ; to access parameters on the stack
        mov      ax,asc_in       ;
        sub      ax,asc_out
        pop      bp
        ret                      ;
_Buf_stat ENDP

;
;
; Procedure Receive_Byte removes characters from the interrupt handler's
; ring buffer & increments the buffer pointer appropriately. Returns
; character in AL.
PUBLIC _Receive_Byte
_Receive_Byte PROC    FAR
        push     bp              ; preserve bp in order to use the bp
        mov      bp,sp           ; to access parameters on the stack
        push     bx              ; save all necessary registers
no_chr: mov       bx,asc_out      ; if no char waiting,loop until one is
received
        cmp      bx,asc_in       ;
        je       no_chr          ;
        xor      ax,ax           ; clear the accumulator
        mov      al,[bx+_Asc_buf] ; store the received byte in AL
        inc      bx              ;
        cmp      bx,Buf_len      ;
        jne     yes_chr          ;
        xor     bx,bx            ; reset ring pointer
yes_chr: mov    asc_out,bx       ; store updated pointer
        pop     bx               ; restore all registers
        pop     bp
        ret                      ;
_Receive_Byte ENDP

;
;
; Procedure Send_Byte sends one character through the port.
PUBLIC _Send_Byte
_Send_Byte PROC    FAR
        push    bp               ; preserve bp in order to use the bp
        mov     bp,sp            ; to access parameters on the stack
        push    ax
        push    dx
wait_loop:
        IPort   Port_C
        and     al,Data_out
        jz      wait_loop
        call    out_delay        ; Used as a necessary delay
        call    out_delay        ; Used as a necessary delay
        mov     ax,[bp+6]
        OPort   Port_B,al        ; transmit the character byte
        pop     dx               ; restore registers and return
        pop     ax
        pop     bp
        ret
_Send_Byte ENDP

;
;
; Procedure Receive_word removes 2 bytes from the interrupt handler's ring
; buffer and increments the buffer pointer appropriately. Returns word in
; AX.
PUBLIC _Receive_Word
_Receive_Word PROC    FAR
        push    bp               ; preserve bp in order to use the bp
        mov     bp,sp            ; to access parameters on the stack
        push    bx               ; save all necessary registers
        push    cx               ;
        push    dx               ;
        mov     cx,2             ; counter for 2 bytes
nothng:
        mov      bx,asc_out       ; if no char waiting,loop until one is
received
        cmp     bx,asc_in        ;
        je      nothng           ;
        mov     ah,[bx+_Asc_buf] ; store the received byte in AH
        inc     bx               ;
        cmp     bx,Buf_len       ;
        jne     yes_wrap         ;
        xor     bx,bx            ; reset ring pointer
yes_wrap: mov   asc_out,bx       ; store updated pointer
        dec     cx               ; decrement byte counter
        jz      done             ; second byte received
        mov     al,ah            ; transfer the first byte into LSB
        jmp     nothng           ; wait for second byte
done:
        pop     dx               ; restore all registers
        pop     cx               ;
        pop     bx                      ;
        pop     bp
        ret                             ;
_Receive_Word ENDP

;
;
; Procedure Send_Word sends 2 bytes through the port. The LSB is sent
; first then the MSB is sent.
PUBLIC _Send_Word
_Send_Word PROC    FAR
        push    bp              ; preserve bp in order to use the bp
        mov     bp,sp           ; to access parameters on the stack
        push    ax
        push    dx
wait_here:
        IPort   Port_C
        and     al,Data_out
        jz      wait_here
        call    out_delay       ; Used as a necessary delay
        call    out_delay       ; Used as a necessary delay
        mov     ax,[bp+6]
        OPort   Port_B,al       ; transmit the LSB
wait_agn:
        IPort   Port_C
        and     al,Data_out
        jz      wait_agn
        call    out_delay       ; Used as a necessary delay
        call    out_delay       ; Used as a necessary delay
        mov     ax,[bp+6]
        OPort   Port_B,ah       ; transmit the MSB
        pop     dx              ; restore registers and return
        pop     ax
        pop     bp
        ret
_Send_Word ENDP

_TEXT ENDS
;
;       The data segment
;
_DATA SEGMENT word public 'DATA'
asc_in    DW    0               ; input pointer to ring buffer
asc_out   DW    0               ; output pointer to ring buffer
intc_oifs DW    0               ; original content of IRQ5
intc_seg  DW    0               ; service vector
      PUBLIC _Asc_buf
_Asc_buf    DB Buf_len DUP(?)
_DATA    ENDS
END
/* end file cint.asm */
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
=    Name              :  ZENON SLODKI                            =
=    DATE              :  April 16, 1990                          =
=    Description       :  ISDN application software               =
=    File name         :  teisdn.c                                =
=    Version           :  2.0                                     =
=    Function          :  PC XT server for the TE                 =
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = */
/* Based on DGM&S sample.c program. */

/* Include files */
/* for AT */
#include    "c:\c\include\stdio.h"
#include    "c:\c\include\sys\types.h"
#include    "c:\c\include\sys\stat.h"
#include    "c:\c\include\io.h"
#include    "c:\c\include\conio.h"
#include    "c:\bin\dgms\te\app.h"

#define     LF            printf("\n")
#define     FALSE         0
#define     TRUE          1
#define     NIL           0

/* ISDN TE modes */
#define          STAT_REQ               0
#define          EXT_REQ                1
#define          BUF_REQ                2

#define          BURST_SIZE             100
#define          ZERO                   0
#define          NINE                   9
#define          FOUR                   4

/* Global variables */
int  status = 0;
int  crn_voice = 0;
int  crn_data = 0;
int  crn_second = 0;
int  crn_file = 0;
int  main_choice = 0;
int key_talk = 0;
unsigned char bfr[BURST_SIZE+5];

/* External assembly language global variables */
extern char Asc_buf[9000];
extern void Install(void);
extern void UN_Install(void);
extern void Send_Byte(int);
extern unsigned char Receive_Byte(void);
extern void Send_Word(int);
extern int Receive_Word(void);

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function          :  main                                    -
-    Input             :  none                                    -
-    Output            :  none                                    -
-    Date              :  19 Oct 89                               -
-    UpDate            :  30 Mar 90                               -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays a menu from which the user selects an ISDN application. */
main()
```

```
{
unsigned char lowb;

/* install device driver and program 8255 PPI */
     Install();
     printf("Installation is complete.\n");

     while( TRUE ) {
             if( Buf_stat() ) {
                    if( key_talk )
                    Receive_Talk_Cmd();
                           else
                    Receive_Command();
                    }
             if( key_talk )
                           issue_receive_immed( crn_second );
     }
} /*end main*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function          :   issue_isdn_transmit                     -
-    Input             :   call reference number                   -
-    Output            :   status                                  -
-    Date              :   19 Oct 89                               -
-    UpDate            :   06 Apr 90                               -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Transmits the buffer of characters entered from the user interface. */
int issue_isdn_transmit( int id )
{
unsigned int bfr_lgth;
unsigned char bfr[BURST_SIZE];
int i;

     bfr_lgth = Receive_Word();
     printf("Received buffer length: %d\n",bfr_lgth);
/* receive buffer characters until end of string character arrives */
     i = 0;
     while( (bfr[i] = Receive_Byte()) != NIL) {
           printf("%c ",bfr[i]);
           i++;
     } /*end while*/

     status = isdn_transmit(id, &bfr[0], bfr_lgth);

     return(status);
} /*end issue_isdn_transmit*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function          :   issue_isdn_receive                      -
-    Input             :   call reference number                   -
-    Output            :   status                                  -
-    Date              :   19 Oct 89                               -
-    UpDate            :   02 Apr 90                               -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Receives the buffer of characters sent through the ISDN interface. */
int issue_isdn_receive( int id )
{
unsigned bfr_lgth,i,j;

     status = isdn_receive_wait(id, &bfr[0], MX_APP_DATASZ);
     bfr_lgth = status + 1;
       /* to include the /0 character */
```

```
      if( status > 0 ) {
          bfr[status] = '\0';
          printf("\tReceived : %-50s\n", &bfr[0]);

          Send_Word(bfr_lgth);
          Send_Buf(bfr);
      }
      else {
          printf("Error - Receive Data : %d\n", status);
          Send_Word(status);
      }

       status = 0;
       return(status);
} /*end issue_isdn_receive*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-     Function          :    issue_file_transmit                    -
-     Input             :    call reference number                  -
-     Output            :    none                                   -
-     Date              :    26 Oct 89                              -
-     UpDate            :    16 Apr 90                              -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Sends the buffer of characters corresponding to the file sent. */
int issue_file_transmit( int id )
{
unsigned int bfr_lgth;
int i;

      bfr_lgth = Receive_Word();
      printf("Received bfr length: %u\n",bfr_lgth);
      while( bfr_lgth == BURST_SIZE ) {
            for(i=0; i<bfr_lgth; i++) {
                  bfr[i] = Receive_Byte();
                  printf("%c",bfr[i]);
            } /*end for*/

            status = isdn_transmit(id, &bfr[0], bfr_lgth);
            bfr_lgth = Receive_Word();
              printf("Received bfr length: %u\n",bfr_lgth);
      } /*end while*/

      for(i=0; i<bfr_lgth; i++) {
             bfr[i] = Receive_Byte();
             printf("%c",bfr[i]);
      } /*end for*/

      status = isdn_transmit(id, &bfr[0], bfr_lgth);

      if( status < 0 )
          printf("Error - Transmit FILE TR^NSFER : %d\n", status);
      return( status );
} /*end issue_file_transmit*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-     Function          :    issue_file_receive                     -
-     Input             :    call reference number                  -
-     Output            :    none                                   -
-     Date              :    25 Oct 89                              -
-     UpDate            :    12 Apr 90                              -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Receives the characters of a file sent through the ISDN interface. */
```

```
int  issue_file_receive( int id )
{
unsigned bytecount,i;
int  j=0;
int  cnta,cntb;

      bytecount = isdn_receive_wait(id, &bfr[0], BURST_SIZE);
      if( bytecount < 0 )
         printf("Error - Receive FILE TRANSFER : %d\n", status);

      while( bytecount == BURST_SIZE ) {
            j++;
         /* send length of received buffer */
         Send_Word( bytecount );

         for(i=0; i<bytecount; i++) {
              putc( bfr[i],stdout );
              Send_Byte( bfr[i] );
         }
                 for(cnta=0; cnta<5555; cnta++)
                      for(cntb=0; cntb<29; cntb++)
                           ;
           bytecount = isdn_receive_wait(id, &bfr[0], BURST_SIZE);

             Send_Byte(BUF_REQ);
      }

      /* send length of received buffer */
      Send_Word( bytecount );
      for(i=0; i<bytecount; i++)  {
         putc( bfr[i],stdout );
         Send_Byte( bfr[i] );
      }

       status = 0;
       return( status );
} /*end issue_file_receive*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   issue_receive_immed                       -
-    Input            :   call reference number                     -
-    Output           :   none                                      -
-    Date             :   01 Feb 90                                 -
-    UpDate           :   02 Feb 90                                 -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Receives data if available through the ISDN interface without waiting.
*/
issue_receive_immed( int id )
{
int status,bytecount,i;
unsigned char bfr[90];

      bytecount = isdn_receive_immed(id, &bfr[0], 90);
      if( bytecount < 0 )  {
         status = isdn_disconnect(crn_second,0);
         crn_second = 0;
         key_talk = 0;
      }
      if( bytecount ) {                        /*one or more bytes*/
         /* send length of received buffer */
         Send_Word( bytecount );
         for(i=0; i<bytecount; i++) {
            putc( bfr[i],stdout );
            Send_Byte( bfr[i] );
        } /*end for*/
    } /*end if*/
} /*end issue_receive_immed*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   Receive_Command                                  -
-    Input           :   ISDN command byte                                -
-    Output          :   none                                             -
-    Date            :   18 Dec 89                                        -
-    UpDate          :   02 Feb 90                                        -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                                      */
Receive_Command()
{
unsigned char val;
unsigned char param;
unsigned int numval;

     /* receive command */
     val = Receive_Byte();

     /* The following commands require a parameter of size BYTE */
     if( (val == 's') || (val == 't') )
         param = Receive_Byte();
     /* The following commands require a parameter of size WORD */
     else if( (val == 'u') || (val == 'v') )
         numval = Receive_Word();

     /* perform the action specified by the command */
     switch (val) {
            case 'a':
                        printf("Voice Dial mode: %c\n",val);
                        status = isdn_dial('\5', "546", "V", uLAW);
                        crn_voice = status;
                        /* send ISDN TE mode */
                        Send_Byte( STAT_REQ );
                        break;
            case 'b':
                        printf("Voice Disconnect mode: %c\n",val);
                        status = isdn_dial_disconnect("W");
                        crn_voice = 0;
                        /* send ISDN TE mode */
                        Send_Byte( STAT_REQ );
                        break;
            case 'c':
                        printf("Voice Reject mode: %c\n",val);
                        status = isdn_voice_reject();
                        crn_voice = 0;
                        /* send ISDN TE mode */
                        Send_Byte( STAT_REQ );
                        break;
            case 'd':
                        printf("Data 64K connect mode: %c\n",val);
                        status = isdn_connect(CALL_TYPE_TRANS,CHANNEL_B,
                        ISDN_PLAN,"654",RATE_64K,"D");
                        crn_data = status;
                        /* send ISDN TE mode */
                        Send_Byte( STAT_REQ );
                        break;
```

```
case 'e':
                printf("Data 64K disconnect mode: %c\n",val);
                status = isdn_disconnect(crn_data,"E");
                crn_data = 0;
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'f':
                printf("Data 64K reject mode: %c\n",val);
                status = isdn_connect_reject(CALL_TYPE_TRANS,0);
                crn_data = 0;
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'g':
                printf("Transmit Data 64K mode: %c\n",val);
                status = issue_isdn_transmit( crn_data );
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'h':
                printf("Receive Wait Data 64K mode: %c\n",val);
                /* send ISDN TE mode */
                Send_Byte( BUF_REQ );
                status = issue_isdn_receive( crn_data );
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'i':
                printf("Key Talk 16K connect mode: %c\n",val);
                status = isdn_connect(CALL_TYPE_TRANS,CHANNEL_B,
                ISDN_PLAN,"754",RATE_16K,0);
                crn_second = status;
                key_talk = 1;
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'j':
                printf("Key Talk 16K disconnect mode: %c\n",val);
                status = isdn_disconnect(crn_second,0);
                crn_second = 0;
                key_talk = 0;
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'k':
                printf("Data 16K reject mode: %c\n",val);
                status = isdn_connect_reject(CALL_TYPE_TRANS,0);
                crn_second = 0;
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'l':
                printf("Talk 16K mode: %c\n",val);
                status = issue_isdn_transmit( crn_second );
                /* send ISDN TE mode */
                Send_Byte( STAT_REQ );
                break;
case 'm':
                printf("Talk 16K mode: %c\n",val);
                status = issue_isdn_receive( crn_second );
                /* send ISDN TE mode */
```

```
            Send_Byte( STAT_REQ );
            break;
case 'n':
            printf("File 64K connect mode: %c\n",val);
            status = isdn_connect(CALL_TYPE_TRANS,CHANNEL_B,
            ISDN_PLAN,"657",RATE_64K,0);
            crn_file = status;
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 'o':
            printf("File 64K disconnect mode: %c\n",val);
            status = isdn_disconnect(crn_file,0);
            crn_file = 0;
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 'p':
            printf("File 64K reject mode: %c\n",val);
            status = isdn_connect_reject(CALL_TYPE_TRANS,0);
            crn_file = 0;
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 'q':
            printf("File Transfer 64K mode: %c\n",val);
            status = issue_file_transmit( crn_file );
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 'r':
            printf("File Receive 64K mode: %c\n",val);
            /* send ISDN TE mode */
            Send_Byte( BUF_REQ );
            status = issue_file_receive( crn_file );
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 's':
            printf("Mute microphone mode: %u\n",param);
            isdn_mute_mic(param);
            status = 0;
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 't':
            printf("Mute earpiece mode: %u\n",param);
            isdn_mute_ear(param);
            status = 0;
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 'u':
            printf("Volume adjust mode: %u\n",numval);
            isdn_audio_vol(numval);
            status = 0;
            /* send ISDN TE mode */
            Send_Byte( STAT_REQ );
            break;
case 'v':
            printf("ISDN status mode: %u\n",numval);
            status = isdn_status(numval);
```

```
                        /* send ISDN TE mode */
                        Send_Byte( STAT_REQ );
                        break;
                case 'z':
                        printf("Graceful shutdown mode");
                        UN_Install();
                        exit( 0 );
                        /* send ISDN TE mode */
                        Send_Byte( STAT_REQ );
                        break;
                default :       printf("Unknown transmission %d\n",val);
                        break;
        } /*end switch*/

        /* send status information back */
        printf("Returning the status word: %d\n",status);
        Send_Word( status );
} /*end Receive_Command*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-       Function        :       Receive_Talk_Cmd                        -
-       Input           :       ISDN command byte                       -
-       Output          :       none                                    -
-       Date            :       02 Feb 90                               -
-       UpDate          :              90                               -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                                      */
Receive_Talk_Cmd()
{
unsigned char cmd;
unsigned char bfr[90];
unsigned int bfr_lgth;
int i;

        /* receive command */
        cmd = Receive_Byte();

        /* The following directives require a buffer size parameter */
        if( (cmd == 'T') || (cmd == 'Q') ) {
                bfr_lgth = Receive_Word();
                if( bfr_lgth > 0 ) {
                        for(i=0; i<bfr_lgth: i++) {
                                bfr[i] = Receive_Byte();
                                putc(bfr[i],stdout);
                        } /*end for*/
                        status=isdn_transmit(crn_second, &bfr[0], bfr_lgth);
                        if( cmd == 'Q' )
                                key_talk = 0;
                }
                else
                        printf("The bfr_lgth : %d\n", bfr_lgth);
                printf("\n");
        } /*end if*/
        else
                printf("Unknown command : %c\n", cmd);
} /*end Receive_Talk_Cmd */

/* end file teisdn.c */
```

IBM PC ISDN NT Terminal Software Listing

```
/* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
=     Name             :   ZENON SLODKI                            =
=     DATE             :   April 11, 1990                          =
=     Description      :   ISDN application software               =
=     File name        :   appisdnt.c                              =
=     Version          :   2.1                                     =
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = */
/* Based on DGM&S sample.c program. */

/* include files */
/* for AT */
#include    "c:\c\include\stdio.h"
#include    "c:\c\include\sys\types.h"
#include    "c:\c\include\sys\stat.h"
#include    "c:\c\include\io.h"
#include    "c:\c\include\conio.h"
#include    "c:\bin\dgms\te\app.h"

#define     LF                  printf("\n")
#define     FALSE               0
#define     TRUE                1
#define     BURST_SIZE          100

/* data structures */
struct interr {
       int intval;
};
struct charerr {
       unsigned char lowval;
       char highval;
};
union dual {
       struct interr  x;
       struct charerr h;
};

/* global variables */
int   status = 0;
int   crn_voice = 0;
int   crn_data = 0;
int   crn_second = 0;
int   crn_file = 0;
int   main_choice = 0;
union dual errcode;
unsigned char bfr[BURST_SIZE+5];

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-     Function         :   main                                    -
-     Input            :   none                                    -
-     Output           :   none                                    -
-     Date             :   19 Oct 89                               -
-     UpDate           :                                           -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays a menu from which the user selects an ISDN application. */
main()
{
     print_main_menu();
     while( TRUE )
          appmain_process();
} /*end main*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   print_main_menu                            -
-    Input            :   none                                       -
-    Output           :   none                                       -
-    Date             :   19 Oct 89                                  -
-    UpDate           :   01 Feb 90                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays the top level menu selection of ISDN applications.      */
print_main_menu()
{
     printf(" 1  - Voice \n");
     printf(" 2  - Data (64Kbps)\n");
     printf(" 3  - Keyboard Conversation (16Kbps)\n");
     printf(" 4  - File Transfer\n");
     printf(" 5  - Microphone & Earpiece\n");
     printf(" 6  - Status\n");
     printf(" 7  - Read Notice \n");
     printf(" 8  - Read DATA Connection Notice \n");
     printf(" 0  - Exit Application\n");
} /*end print_main_menu*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   print_sub1_menu                            -
-    Input            :   none                                       -
-    Output           :   none                                       -
-    Date             :   19 Oct 89                                  -
-    UpDate           :   31 Oct 89                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays the second level menu selection of ISDN applications.   */
print_sub1_menu()
{
     printf(" 1  - Make Call\n");
     printf(" 2  - Accept Call\n");
     printf(" 3  - Disconnect Call\n");
     printf(" 4  - Reject Call\n");
     printf(" 5  - Send\n");
     printf(" 6  - Receive\n");
     printf(" 0  - Exit to Main Menu\n");
} /*end print_sub1_menu*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   print_sub2_menu                            -
-    Input            :   none                                       -
-    Output           :   none                                       -
-    Date             :   19 Oct 89                                  -
-    UpDate           :   31 Oct 89                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays the second level menu selection of ISDN applications.   */
print_sub2_menu()
{
     printf(" 1  - Mute Microphone\n");
     printf(" 2  - Mute Earpiece\n");
     printf(" 3  - Change Microphone Volume\n");
     printf(" 0  - Exit to Main Menu\n");
} /*end print_sub2_menu*/
```

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   appmain_process                          -
-    Input            :   none                                     -
-    Output           :   none                                     -
-    Date             :   19 Oct 89                                -
-    UpDate           :   31 Oct 89                                -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                              */
appmain_process()
{
     if( kbhit() )
         process_keyboard_main();
} /*end appmain_process*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   appsub_process                           -
-    Input            :   none                                     -
-    Output           :   none                                     -
-    Date             :   19 Oct 89                                -
-    UpDate           :   08 Nov 89                                -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                              */
appsub_process()
{
   process_keyboard_sub();
} /*end appsub_process*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   process_keyboard_main                    -
-    Input            :   none                                     -
-    Output           :   none                                     -
-    Date             :   19 Oct 89                                -
-    UpDate           :   02 Feb 90                                -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                              */
process_keyboard_main()
{

     scanf("%d", &main_choice);
     putch('\n');
     switch( main_choice ) {
         case 0 : exit( 0 ); break;
         case 1 :
                  print_sub1_menu();
                  appsub_process();
                  break;
         case 2 :
                  print_sub1_menu();
                  appsub_process();
                  break;
         case 3 :
                  print_sub1_menu();
                  appsub_process();
                  break;
         case 4 :
                  print_sub1_menu();
                  appsub_process();
                  break;
         case 5 :
                  print_sub2_menu();
                  appsub_process();
                  break;
```

```
        case 6 :
                 call_isdn_status();
                 break;
        case 7 :
                 call_isdn_read_notice();
                 break;
        case 8 :
                 call_isdn_read_confirm();
                 break;
        default  : printf("You entered an invalid selection\n"); break;
    }
    print_main_menu();
} /*end process_keyboard_main*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   process_keyboard_sub                       -
-    Input           :   none                                       -
-    Output          :   none                                       -
-    Date            :   19 Oct 89                                  -
-    UpDate          :   02 Feb 90                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -*/
/*                                          */
process_keyboard_sub()
{
unsigned int ch;

     scanf("%d", &ch);
     putch('\n');
     ch = ch + main_choice*10;

     switch( ch ) {
         case 10: case 20: case 30: case 40: case 50: break;
         case 11 :
                   call_isdn_dial();
                   break;
         case 12 :
                   call_isdn_voice_accept();
             /* used just to indicate activity for STATUS function */
                   crn_voice = 16;
                   break;
         case 13 :
                   crn_voice = call_isdn_dial_disconnect();
                   break;
         case 14 :
                   crn_voice = call_isdn_voice_reject();
                   break;
         case 21 :
                   crn_data = call_isdn_connect( RATE_64K );
                   break;
         case 22 :
                   crn_data = call_isdn_connect_accept();
                   break;
         case 23 :
                   crn_data = call_isdn_disconnect( crn_data );
                   break;
         case 24 :
                   crn_data = call_isdn_connect_reject();
                   break;
         case 25 :
                   call_isdn_transmit( crn_data );
                   break;
```

```
            case 26 :
                        call_isdn_receive_wait( crn_data );
                        break;
            case 31 :
                        crn_second = call_isdn_connect( RATE_16K );
                        break;
            case 32 :
                        crn_second = call_isdn_connect_accept();
                        break;
            case 33 :
                        crn_second = call_isdn_disconnect( crn_second );
                        break;
            case 34 :
                        crn_second = call_isdn_connect_reject();
                        break;
            case 35 :
                        call_key_talk( crn_second );
                        break;
            case 36 :
                        call_key_talk( crn_second );
                        break;
            case 41 :
                        crn_file = call_isdn_connect( RATE_64K );
                        break;
            case 42 :
                        crn_file = call_isdn_connect_accept();
                        break;
            case 43 :
                        crn_file = call_isdn_disconnect( crn_file );
                        break;
            case 44 :
                        crn_file = call_isdn_connect_reject();
                        break;
            case 45 :
                        call_file_transmit( crn_file );
                        break;
            case 46 :
                        call_file_receive_wait( crn_file );
                        break;
            case 51:
                        call_isdn_mute_mic();
                        break;
            case 52:
                        call_isdn_mute_ear();
                        break;
            case 53:
                        call_isdn_audio_vol();
                        break;
            default : printf("You entered an invalid selection\n"); break;
        } /*end switch*/
} /*end process_keyboard_sub*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :    call_isdn_status                         -
-    Input           :    none                                     -
-    Output          :    none                                     -
-    Date            :    31 Oct 89                                -
-    UpDate          :    10 Jan 90                                -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Display the status of all connections.     */
call_isdn_status()
{
      if( crn_voice  == 0 )
          printf("Voice connection is INACTIVE.\n");
      else
          printf("Voice connection may be ACTIVE.\n");

      if( crn_data == 0 )
          printf("Data (64 Kbps) connection is INACTIVE.\n");
      else {
          status = isdn_status(crn_data);
          if( status > 0 )
              printf("Data (64 Kbps) connection is ACTIVE.\n");
          else {
              printf("Data (64 Kbps) connection is INACTIVE.\n");
          /* update crn information */
              crn_data = 0;
          }
      }

      if( crn_second == 0 )
          printf("Second data (16 Kbps) connection is INACTIVE.\n");
      else {
          status = isdn_status(crn_second);
          if( status > 0 )
              printf("Second data (16 Kbps) connection is ACTIVE.\n");
          else {
              printf("Second data (16 Kbps) connection is INACTIVE.\n");
          /* update crn information */
              crn_second = 0;
          }
      }

      if( crn_file == 0 )
          printf("File transfer data connection is INACTIVE.\n");
      else {
          status = isdn_status(crn_file);
          if( status > 0 )
          printf("File transfer data connection is ACTIVE.\n");
          else {
              printf("File transfer data connection is INACTIVE.\n");
          /* update crn information */
              crn_file = 0;
          }
      }
} /*end call_isdn_status*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   call_isdn_read_notice                     -
-    Input            :   none                                      -
-    Output           :   none                                      -
-    Date             :   02 Feb 90                                 -
-    UpDate           :          90                                 -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays any user information fields that are available. */
call_isdn_read_notice()
{
char bfr[90];

       status = isdn_read_notice(&bfr[0], 90);
       if( status > 0 )
            printf("CODE notice: %c\n",bfr[0]);
       else
            printf("NO CODE notice.\n");
```

```
} /*end call_isdn_read_notice*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function          :   call_isdn_read_confirm                    -
-    Input             :   none                                      -
-    Output            :   none                                      -
-    Date              :   02 Feb 90                                 -
-    UpDate            :          90                                 -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays any user information fields that are available. */
call_isdn_read_confirm()
{
char bfr[90];

      status = isdn_read_confirm(crn_data, &bfr[0], 90);
      if( status > 0 )
            printf("DATA CODE notice: %c\n",bfr[0]);
      else
            printf("NO DATA CODE notice.\n");
} /*end call_isdn_read_confirm*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -   - - -
-    Function          :   Display_status                            -
-    Input             :   none                                      -
-    Output            :   none                                      -
-    Date              :   31 Oct 89                                 -
-    UpDate            :   05 Jan 90                                 -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Displays the interpreted Layer 3 status code.
   The error codes are not supported by DGM&S. Function not implemented.*/
/*
Display_status( int value )
{
printf("Display status: %X\n",-value);

errcode.x.intval = -value;
  switch( -errcode.h.highval ) {
     case -1 : printf("Invalid parameter error\n"); break;
     case -2 : printf("Parameter conflict error\n"); break;
     case -3 : printf("Illogical event error\n"); break;
     case -4 : printf("Connection busy\n"); break;
     case -5 : printf("Connection failed error\n"); break;
     case -6 : printf("Disconnected\n"); break;
     case -7 : printf("Aborted\n"); break;
     case -8 : printf("Shared memory send failed error\n"); break;
     case -9 : printf("Shared memory receive failed error\n"); break;
     case -10: printf("X.25 reference number not found\n"); break;
     case -11: printf("X.25 reference number in use\n"); break;
     case -12: printf("X.25 multi-frame not established\n"); break;
     case -13: printf("X.25 window size exceeded\n"); break;
     default :
     printf("Highval error parameter not found:
       %X\n",errcode.h.highval);break;
  }

  switch( errcode.h.lowval ) {
     case 16 : printf("Normal\n"); break;
     case 17 : printf("User busy\n"); break;
     case 18 : printf("No user responding\n"); break;
     case 21 : printf("Call rejected\n"); break;
     case 22 : printf("Number changed error\n"); break;
     case 25 : printf("Call resumed\n"); break;
     case 26 : printf("Invalid destination address error\n"); break;
     case 29 : printf("Requested facility rejected\n"); break;
     case 33 : printf("Circuit out of order error\n"); break;
     case 34 : printf("No channel available error\n"); break;
     case 35 : printf("Destination not obtainable error\n"); break;
     case 42 : printf("Network congested error\n"); break;
     case 50 : printf("Requested facility not subscriber error\n"); break;
     case 54 : printf("Incoming calls barred error\n"); break;
     case 65 : printf("Bearer service not implemented error\n"); break;
     case 66 : printf("Channel type not implemented error\n"); break;
     case 68 : printf("Message not implemented error\n"); break;
     case 69 : printf("Requested facility not implemented error\n");
                       break;
     case 81 : printf("Invalid call reference number error\n"); break;
     case 82 : printf("ID channel does not exist error\n"); break;
     case 85 : printf("Digit is invalid error\n"); break;
     case 88 : printf("Incompatible destination error\n"); break;
     case 91 : printf("Transmitting network does not exist error\n");
                       break;
     case 93 : printf("Mandatory missing error\n"); break;
     case 97 : printf("Message is bad or not implemented error\n"); break;
     case 98 : printf("Message is bad in call state error\n"); break;
     case 99 : printf("Information element bad or not implemented\n");
                       break;
     case 100: printf("Bad information element error\n"); break;
     case 127: printf("Cause unknown error\n"); break;
     case 255: printf("TEI removed error\n"); break;
     default :
     printf("Lowval error parameter not found:
             %X\n",errcode.h.lowval);break;
  }
} /end Display_status/
*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   call_isdn_dial                            -
-    Input            :   none                                      -
-    Output           :   call reference number                     -
-    Date             :   19 Oct 89                                 -
-    UpDate           :   05 Jan 90                                 -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Make an outgoing voice connection. This function is not supported
   on an NT workstation. */
int call_isdn_dial()
{
   printf("The NT workstation cannot initiate a voice connection.\n");
} /*end call_isdn_dial*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   call_isdn_voice_accept                    -
-    Input            :   none                                      -
-    Output           :   none                                      -
-    Date             :   24 Oct 89                                 -
-    UpDate           :          89                                 -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Accept an incoming voice connection.     */
call_isdn_voice_accept()
{
static unsigned char volume;
static unsigned char routing;
```

```
    /* set default volume at level 5 */
    volume = 5;
    /* set default routing for hadset since CPU option not available */
    routing = 1;

    status = isdn_voice_accept(volume, routing);
    if( status < 0 )
        printf("Error - Accept Voice Call : %d\n", status);
} /*end call_isdn_voice_accept*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   call_isdn_dial_disconnect                  -
-    Input            :   none                                       -
-    Output           :   clear call reference number                -
-    Date             :   19 Oct 89                                  -
-    UpDate           :   31 Oct 89                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                              */
int call_isdn_dial_disconnect()
{
    status = isdn_dial_disconnect(0);
    if( status < 0 )
        printf("Error - Disconnect Voice Call : %d\n", status);
    return(0);
} /*end call_isdn_dial_disconnect*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   call_isdn_voice_reject                     -
-    Input            :   none                                       -
-    Output           :   clear call reference number                -
-    Date             :   31 Oct 89                                  -
-    UpDate           :          89                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Reject an incoming voice connection.     */
int call_isdn_voice_reject()
{
    status = isdn_voice_reject();
    if( status < 0 )
        printf("Error - Reject Voice Call : %d\n", status);
    return(0);
} /*end call_isdn_voice_reject*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   call_isdn_connect                          -
-    Input            :   rate adaption                              -
-    Output           :   call reference number                      -
-    Date             :   19 Oct 89                                  -
-    UpDate           :   05 Jan 90                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Make an outgoing data connection. This function is not supported
   on an NT workstation. */
int call_isdn_connect( int rate )
{
   printf("The NT workstation cannot initiate a data connection.\n");
} /*end call_isdn_connect*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_isdn_connect_accept                    -
-    Input           :   none                                        -
-    Output          :   call reference number                       -
-    Date            :   19 Oct 89                                   -
-    UpDate          :   31 Oct 89                                   -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                          */
int call_isdn_connect_accept()
{
     status = isdn_connect_accept(CALL_TYPE_TRANS, 0, 0);
     if( status < 0 )
         printf("Error - Data Call Accept : %d\n", status);
     return( status );
} /*end call_isdn_connect_accept*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_isdn_connect_reject                    .
-    Input           :   none                                        -
-    Output          :   clear call reference number                 -
-    Date            :   19 Oct 89                                   -
-    UpDate          :   31 Oct 89                                   -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                          */
int call_isdn_connect_reject()
{
     status = isdn_connect_reject(CALL_TYPE_TRANS, 0);
     if( status < 0 )
         printf("Error - Data Call Reject : %d\n", status);
     return(0);
} /*end call_isdn_connect_reject*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_isdn_disconnect                        -
-    Input           :   call reference number                       -
-    Output          :   clear call reference number                 -
-    Date            :   19 Oct 89                                   -
-    UpDate          :   31 Oct 89                                   -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                          */
int call_isdn_disconnect( int id )
{
     status = isdn_disconnect(id, 0);
     if( status < 0 )
         printf("Error - Disconnect Data Call : %d\n", status);
     return(0);
} /*end call_isdn_disconnect*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_key_talk                               -
-    Input           :   call reference number                       -
-    Output          :   none                                        -
-    Date            :   01 Feb 90                                   -
-    UpDate          :   02 Feb 90                                   -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Transmits the buffer of characters entered fromthe user interface.
     Connection is terminated when the user enters ".q" meaning quit. */
call_key_talk( int id )
{
unsigned char     active_flag=1;
unsigned char     line[90];
int                     status,i;
```

```
      while( active_flag ) {
            if( kbhit() ) {
                  gets(line);
                  if( (line[0] == '.') && (line[1] == 'q') ) {
                        status = isdn_transmit(id,"Conversation
                                 Over.\n\0",20);
                        crn_second = call_isdn_disconnect( id );
                        active_flag = 0;
                  }
                  else if( (line[0] == '.') && (line[1] == 'e') )
                        active_flag = 0;
                  else
                        status = isdn_transmit(id, &line[0],
                                 strlen(&line[0]) );
            } /*end if*/

            /* no keyboard input */
            status = isdn_receive_immed(id, &line[0], 90);
            if( status < 0 )
            /* connection has been lost */
                  call_isdn_disconnect( id );
            else if( status ) {
                  for(i=0; i<status; i++)
                        putc(line[i], stdout);
                  printf("\n");
            }
      } /*end while*/
} /*end call_key_talk*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-     Function        :   call_isdn_transmit                         -
-     Input           :   call reference number                      -
-     Output          :   none                                       -
-     Date            :   19 Oct 89                                  -
-     UpDate          :   10 Jan 90                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                               */
call_isdn_transmit( int id )
{
unsigned int bfr_lgth;
char temp;

     scanf("%c",&temp); /* clear buffer */
     printf("Please enter data\n");
     gets(bfr);

/*     bfr_lgth = min( strlen(&bfr[0]), BURST_SIZE ); */
/*      (ROURKE) assuming that min is a macro, I will replace this line
        with the following code: */
        if ( strlen(&bfr[0]) < BURST_SIZE ) bfr_lgth = strlen(&bfr[0]);
        else bfr_lgth = BURST_SIZE;

     status = isdn_transmit(id, &bfr[0], bfr_lgth);
     if( status < 0 )
         printf("Error - Transmit Data : %d\n", status);
} /*end call_isdn_transmit*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function          :   call_file_transmit                       -
-    Input             :   call reference number                    -
-    Output            :   none                                     -
-    Date              :   26 Oct 89                                -
-    UpDate            :   31 Oct 89                                -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                                    */
call_file_transmit( int id )
{
static unsigned int numread;
FILE *fp;                                    /* file pointer to open file */
char filename[81];                           /* name of file to open */
char done;

     scanf("%c",&done); /* clear buffer */
     printf("Please enter the name of the file to transmit: ");
     gets(filename);

     while( (fp=fopen(filename, "r")) == NULL) {
          printf("Error opening file: %s\n",filename);
          printf("Please enter the name of the file to transmit: ");
          gets(filename);
     } /*end while*/

     done = FALSE;
     while( !done) {
          numread = fread( (void*)bfr,sizeof(char),BURST_SIZE,fp);

          status = isdn_transmit(id, &bfr[0], numread);
          if( status < 0 )
              printf("Error - Transmit FILE TRANSFER : %d\n", status);
          if( numread < BURST_SIZE )
              if( feof(fp) != 0 )
                  done = TRUE;
     } /*end while*/
     fclose(fp);
} /*end call_file_transmit*/


/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function          :   call_isdn_receive_wait                   -
-    Input             :   call reference number                    -
-    Output            :   none                                     -
-    Date              :   19 Oct 89                                -
-    UpDate            :   31 Oct 89                                -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                                    */
call_isdn_receive_wait( int id )
{

     status = isdn_receive_wait(id, &bfr[0], BURST_SIZE);
     if( status > 0 ) {
         bfr[status] = '\0';
         printf("\tReceived : %-50s\n", &bfr[0]);
     }
     else if( status < 0 )
         printf("Error - Receive Data : %d\n", status);
} /*end call_isdn_receive_wait*/
```

```
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_file_receive_wait                     -
-    Input           :   call reference number                      -
-    Output          :   none                                       -
-    Date            :   25 Oct 89                                  -
-    UpDate          :   11 Apr 90                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                                */
call_file_receive_wait( int id )
{
int bytecount,i;

     bytecount = isdn_receive_wait(id, &bfr[0], BURST_SIZE);
     if( bytecount < 0 )
         printf("Error - Receive FILE TRANSFER : %d\n", status);

     while( bytecount == BURST_SIZE ) {
         for(i=0; i<bytecount; i++)
             putc( bfr[i],stdout );
         bytecount = isdn_receive_wait(id, &bfr[0], BURST_SIZE);
     } /*end while*/

     for(i=0; i<bytecount; i++)
         putc( bfr[i],stdout );
} /*end call_file_receive_wait*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_isdn_audio_vol                        -
-    Input           :   none                                       -
-    Output          :   none                                       -
-    Date            :   19 Oct 89                                  -
-    UpDate          :                                              -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                                */
call_isdn_audio_vol()
{
static unsigned int volume;

     printf("Please enter volume (0-9)\n");
     scanf("%d", &volume);

     status = isdn_audio_vol(volume);
     if( status < 0 )
         printf("Error - Adjust Volume : %d\n", status);
} /*end call_isdn_audio_vol*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function        :   call_isdn_mute_mic                         -
-    Input           :   none                                       -
-    Output          :   none                                       -
-    Date            :   19 Oct 89                                  -
-    UpDate          :   24 Oct 89                                  -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                                */
call_isdn_mute_mic()
{
static unsigned char mute;

     printf("Please enter mute mic (0=ON - 1=OFF)\n");
     scanf("%d", &mute);

     status = isdn_mute_mic(mute);
     if( status < 0 )
         printf("Error - Mute Microphone : %d\n", status);
} /*end call_isdn_mute_mic*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-    Function         :   call_isdn_mute_ear                       -
-    Input            :   none                                     -
-    Output           :   none                                     -
-    Date             :   31 Oct 89                                -
-    UpDate           :          89                                -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                          */
call_isdn_mute_ear()
{
static unsigned char mute;

     printf("Please enter mute ear (0=ON - 1=OFF)\n");
     scanf("%d", &mute);

     status = isdn_mute_ear(mute);
     if( status < 0 )
         printf("Error - Mute Earpiece : %d\n", status);
} /*end call_isdn_mute_ear*/
/* end file appisdnt.c */
```