CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

FAST PROTOTYPING AND VALIDATION OF COMMUNICATIONS PROTOCOLS

A MASTER'S THESIS
SUBMITTED TO THE
SCHOOL OF GRADUATE STUDIES AND RESEARCH
OF THE UNIVERSITY OF OTTAWA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE
IN SYSTEMS SCIENCE

By
SHAHID H. MIR

August, 1983

CONTENTS

ACKNOWLEDGEMENTS

The author wishes to express his thanks and deep gratitude to Dr. Robert L. Probert for his guidance, suggestions and valuable supervision.

Thanks are also due to Mr. H. Ural for his helpful suggestions throughout the preparation of this thesis. The author also wishes to express his thanks to Mr. D. Lefebvre for his expert advice and help in the use of the RSX-11M system directives and for his programming suggestions.

The author also wishes to offer his special thanks to his wife Mrs. Naila Mir for her patience and moral support throughout the preparation of this thesis.

ABSTRACT

Recent increases in the complexity of computer networks have necessitated the use of protocols of increased complexity. The reliability of a data communications network depends heavily on the reliability of the protocols involved. Various methods for modelling and testing protocols have been proposed in the literature [9, 10, 12]. Here, we represent the ISO Transport Service specifications in the form of executable specifications. These specifications have been described as a combination of abstract data type technique and a finite state automaton approach, [22, 23] . We also represent the ISO Class-0 Transport Protocol specifications in the form of executable specifications. The Class-0 Tranport Protocol specifications are described using a finite state automaton approach, [24] The basic philosophy behind developing the executable specifications is that of Fast Prototyping. A Prototype is a small version of the ultimate system. It may be an inexpensive subset of the whole system which does not necessarily have all system features but allows the user a hands-on experience to establish his requirements. A software prototype can be developed by using a high level programming language to directly implement the specifications of the intended system as executable specifications [19] . Prototypes of Transport Service specifications and Class-0 Transport Protocol specifications are developed on the PDP 11/34 under RSX-11M using a popular programming language (PASCAL). Representative test sequences were applied to the prototype for Class-0 Transport Protocol specifications and the results are summarized.

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

With advancements in computer communications and software engineering, the complexity of communications software is increasing very rapidly. Complex computer communications networks give rise to a complicated set of rules for exchanging data among a number of hosts. This set of rules form the communication protocols.

The increasing complexity of protocols makes their use more prone to errors, often resulting in erroneous or ambiguous states of data exchange. This necessitates the application of software validation techniques to communications software.

Testing architectures which can be used to test protocols or their prototypes at local (laboratory) or remote (client) sites have been proposed in the literature [20, 21]. It is imperative to develop prototypes of various services and protocols which can represent the specifications in the most accurate manner in a commonly used programming language.

Structuring and standardization of these services and protocols has become a major concern to experts in the communications area. International organizations like CCITT

(International Telegraphy and Telephony Consultative Committee) and ISO (International Standards Organization) have expended much time and money in this Standardization process.

A very brief history of computer networks is presented in Chapter 2 leading to the concept of Layered Architecture of computer networks. The ISO Reference Model for Open System Interconnection is presented very briefly and comments are made on parallel standardization activities by CCITT. Some basic definitions related to communications networks are presented in Chapter 2. A section of this chapter deals with an introduction to the basic concepts of fast prototyping. Existing methods of formal description of protocol specifications are described [11]. A survey of existing protocol validation and verification techniques is presented. A section of this chapter discusses the applicability of current software development and testing concepts to validation of protocols.

Chapter 3 deals with the application of a fast prototyping approach to developing prototypes for Transport Service specifications and Class-0 Transport Protocol specifications. ISO Transport Service specifications and Class-0 Transport Protocol specifications are presented, [22, 23, 24] An executable prototype for the Transport Service specifications is developed as a first step. The Transport Service specifications used for this purpose are described as a

combination of the abstract data type technique and finite state automaton approach, [22, 23]. Then an executable prototype is developed for Class-0 Transport Protocol specifications as a second step. Class-0 Transport Protocol specifications used for this purpose are described on a finite state automaton based approach, [24]. Experience gained by developing the prototype for Transport Service specifications is utilized in modelling the Network Service Provider consisting of two Network Entities and two queues. It is then included as the Network Service Provider for the prototypes of the Class-0 Transport Protocol specifications. Thus, the impact of the layered network architecture of ISO OSI Reference Model is also highlighted in the development of the prototype for Class-0 Transport Protocol specifications. A popular high level programming language (PASCAL) is used for development of the prototypes of the Transport Service specifications, Class-0 Transport Protocol specifications and the model of the Network Service Provider. Software design considerations and quality assurance issues are presented in section 3.5.

Chapter 4 deals with the application of fast prototypes developed in chapter 3 to related research in protocol testing. This chapter presents a brief survey of the protocol testing architectures proposed in the literature, role of fast prototypes in these architectures, the plan developed to exercise the fast prototype for Class-0 Transport Proto-

col specifications in an elementary configuration for laboratory testing of protocols and the results of that exercise.

A summary, conclusions and suggestions for further research is presented in Chapter 5.

Chapter 2

SURVEY OF EXISTING TECHNIQUES FOR PROTOCOL DESIGN AND TESTING

This chapter presents the concept of layered network architecture, an overview of the ISO OSI Reference Model and parallel standardization efforts by CCITT. Some basic definitions are given. An introduction to the basic concepts of fast prototyping is given. A survey of the current protocol validation techniques is presented and relation of some software testing techniques to protocol testing is presented.

2.1 CONCEPT OF LAYERED NETWORK ARCHITECTURE

Earlier design of Computer Networks was mainly on an ad-hoc experimental basis [2]. There was not a well defined and precise set of guidelines. Networks were designed according to the individual organization's requirements. This led to a variety of problems when it came to connecting two different networks together. Consequently, International organizations like ISO and CCITT became involved in developing standards for Computer Networks design.

Today networks are organized as a hierarchy of layers, often denoted "layered network architecture". ISO has come

up with a layered model of a network for Open System Interconnection called the ISO OSI REFERENCE MODEL. This model is defined and discussed in the section 2.2, [3].

Layering is a technique which involves defining the structure of a network as being logically composed of a succession of layers. Each lower layer is isolated from the higher layers. Each layer has its own set of functions to perform and a set of services to provide to its next higher layer. The basic idea of layering is that each layer performs a set of functions (independent of the other layers) and adds value to the services provided by its lower layers in a manner that the total quality of transfer of data is improved.

Taking the example of the ISO Network layer and the Transport Layer, the Network Layer performs the following functions :

1. Establish a Network connection with negotiation,
2. efficient routing of data ,

etc.

This set of functions of the Network Layer is assumed by the Transport Layer in performing its own set of functions which are those necessary to bridge the gap between the services available from the Network Layer and those to be offered to the Transport Service User (i.e Session Layer entity), [21]. The idea is that two peer layers "talk" to

eachother by using the "service" provided to them by the lower layer entities.

One main concept is that the Network Layer plus the layers under it constitute the Network Service Provider and together they provide a set of services to the Network Service user (i.e Transport Layer entity). Now, building on the same concept, when a Transport Layer is added to the Network Service Provider to enhance the quality of service, then the Transport Layer plus the layers below it form a Transport Service Provider which provides transport services to its user (i.e Session Layer entity) at the next layer above the Transport Layer.

This concept is very similar to the underlying concepts of modular programming where the internal working of a module is totally hidden from the other modules and the module performs a service for a calling module. The related idea in Software Engineering is called information hiding [19].

The ISO OSI REFERENCE MODEL consists of seven layers as shown in Fig.2.1. Layering is done according to well defined principles. Justification for a seven layered model can be found in [2, 3, 15, 25]. An overview of these layers can be found in section 2.2.

The standardization efforts of CCITT in this direction are described in section 2.3 and their relation to ISO activities is discussed.

2.2 OVERVIEW OF ISO OSI REFERENCE MODEL

The term Open System Interconnection (OSI) implies that information could be exchanged among systems, computers, networks etc. that are OPEN to one another by virtue of their mutual use of these interface standards. OPENNESS does not imply any particular implementation or some physical means of interconnection of the different entities but refers to the mutual acceptance and implementation of the ISO standards.

The ISO OSI Reference Model consists of seven layers. A brief description of each layer and the services it provides is given below. These descriptions are based on [3, 25].

2.2.1 The Physical Layer

The Physical layer provides the necessary mechanical, electrical or any other physical means of transferring data. It is THE physical link among the hosts.

2.2.2 The Data Link Layer

The Data Link layer converts an unreliable transmission channel into a reliable channel which is used by the Network layer directly above it. It is therefore providing a service to ensure reliability of transfer of data.

2.2.3 The Network Layer

The Network layer is mainly concerned with the routing of data from a source host to a destination host. This layer is responsible for effective routing i.e attempting to avoid congestion. This layer is extremely important in the sense that the efficiency of the network depends on it. The effect of poor routing could be disasterous in a point to point network but the routing problem does not arise in satellite communications where a single channel exists [2]. Packet Switching Public Data Networks (PDN's) have been established in many countries based on the CCITT Recommendation X.25 virtual circuit services. These networks are being interconnected to provide an inter-network data communications facility. CCITT has defined Recommendation X.75 for such connections. Special attention has been given to multiple circuit procedures between communicating Signal Terminating Equipment, which permit virtual calls to simultaneously exist over a number of physical circuits. This allows for load sharing and graceful recovery from circuit failures provided at least one circuit remains operational, [26]

2.2.4 The Transport Layer

The function of the Transport layer is to provide reliable host to host communication for use by the Session layer above it. It hides the details of the communication subnet-

work from the Session layer. The control of data transportation from source host to destination host in an efficient and cost effective way without letting the higher layers know about the details is the job of the Transport layer [1, 25].

More formally, the functions performed by the Transport Layer are at least those necessary to bridge the gap between the services available from the Network layer and those to be provided to the Transport Layer users i.e a Session entity, [23].

The basic functions of the Transport Layer can be categorized as :

1. Transport Connection Establishment phase.
2. Transport Data Transfer phase.
3. Transport Connection Release phase.

A brief description of the Transport Service specifications describing the above functions is presented in section 3.1. Details can be found in [22, 23].

The Transport Protocol has five different classes. Each class defines a set of functions. These classes are listed below with their definitions. Details of each class can be found in [24].

1. Class 0 : Simple class. Class 0 provides the three phases of Transport connection listed above with er-

ror detection and reporting. Recovery from these errors is not part of the Class-0 Transport Protocol specifications.

2. Class 1 : Basic Error Recovery Class. This Class provides the functionality of Class 0 along with the ability to recover after a failure is signalled by the Network Service without involving the Transport Service user.

3. Class 2 : Multiplexing Class. Class 2 provides multiple Transport connections with or without individual flow control. No error detection or error recovery is provided. Multiplexing and demultiplexing allows several transport connections to share a network connection at the same time. Multiplexing allows the concatenation of TPDU's belonging to different transport connections to be transferred in the same Network Data primitive.

4. Class 3 : Error Recovery and Multiplexing Class. Class 3 provides the functionality of Class 2 plus the ability to recover after a failure is signalled by the Network Service without involving the Transport Service user.

5. Class 4 : Error Detection and Multiplexing Class. This class provides the functionality of Class 3 plus the ability to detect and recover from lost, duplicated, or out-of-sequence Transport Protocol Data

Units (TPDU's) without involving the user of the Transport Service.

2.2.5 The Session Layer

The purpose of Session Layer is to assist in the support of the interactions between cooperating Presentation Layer entities. To do this, the Session Layer provides services which can be classified in the following two categories :

1. Binding two presentation entities into a relationship and unbinding them. This is called the Session Administration Service.
2. Control of data exchange, delimiting and synchronizing data operations between two Presentation entities. This is called the Session Dialogue Service [25].

2.2.6 The Presentation Layer

The purpose of this layer is to provide the set of services which may be selected by the Application layer (on top of it) to enable it to interpret the meaning of the data exchanged. These services are for the management of the entry exchange, display and control of structured data. In other words it performs useful transformations on the data to be sent such as text compression [1, 25].

2.2.7 The Application Layer

This is the highest layer in the ISO OSI Reference Model. Protocols of this layer directly serve the end user by providing the distributed information service appropriate to an application, to its management and to system management. Management of Open System Interconnection comprises those functions required to initiate, maintain, terminate and record data concerning the establishment of connections for data transfer among application processes. The other layers exist only to support this layer.

An application is composed of cooperating Application Processes which communicate according to Application Layer protocols. Application processes are the ultimate source and sink for exchanged data, [25].

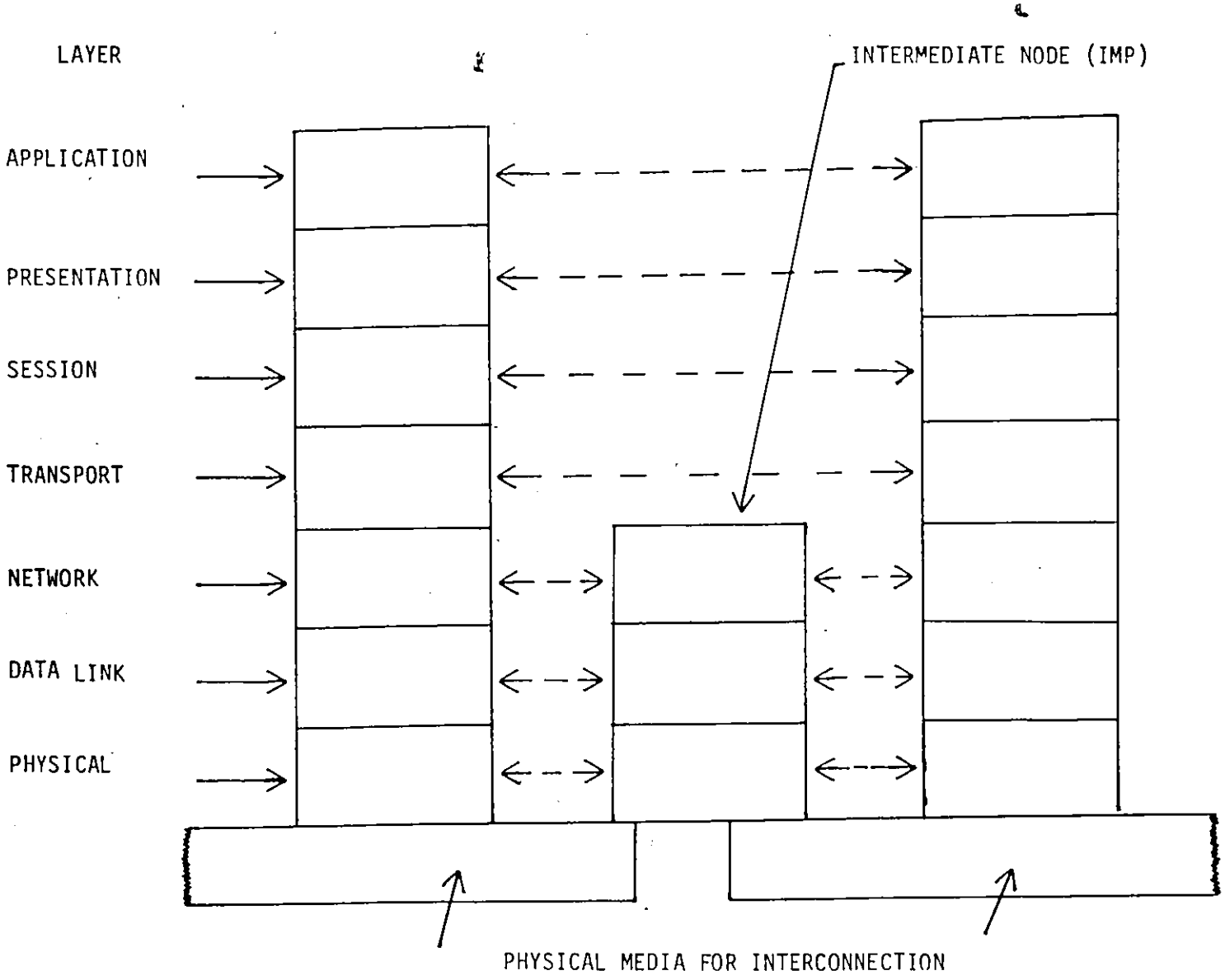


FIG. 2.1

ISO REFERENCE MODEL FOR OPEN SYSTEM INTERCONNECTION

2.3 PARALLEL EFFORTS BY CCITT

The information in this section is based on [2]. CCITT is developing a 'Layered Model of Public Data Networks Service Applications' for the purpose of interconnecting a number of hosts or gaining access to the network services, [2]. ISO has already come up with the OSI Reference Model which has been presented in the previous section. A high degree of compatibility exists between CCITT's and ISO's models which is apparent from Fig.2.2. This compatibility is facilitated by the cooperation which already exists between the two organizations.

Both ISO and CCITT have adhered to the structuring technique of Layering. Both have come up with a seven layered model (see Fig.2.2). The major difference is in the interpretation of the services provided by the Network and Transport layers and their relationship to X.25 virtual circuits. This is also visible in the Fig.2.2. CCITT's view is that the services provided by the Transport and the Network layers are identical except in the quality of service provided. ISO maintains that the services provided by the Network layer are somewhat primitive, resulting in distinctly different services provided by the Network and Transport layers.

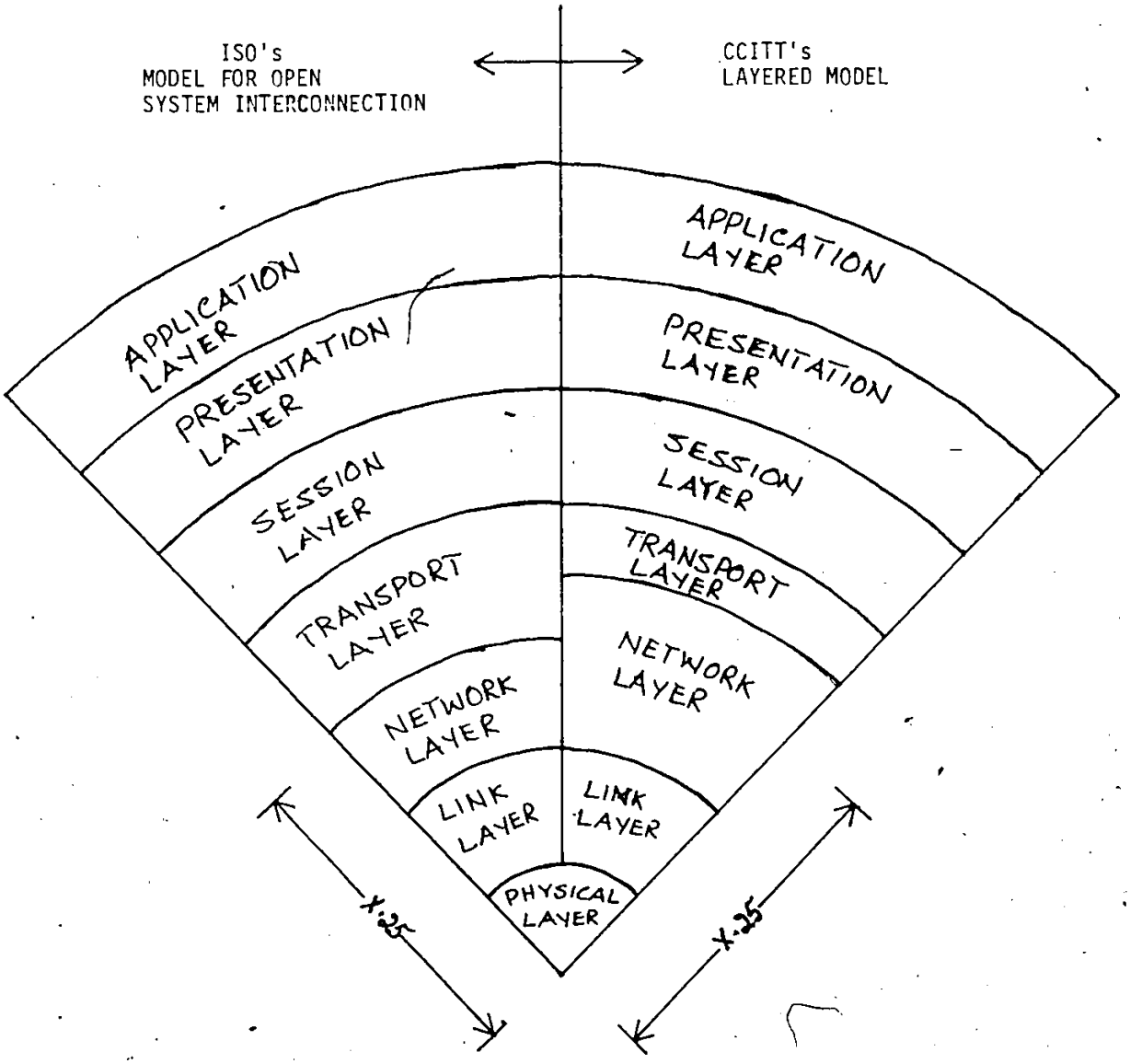


FIG. 2.2

ISO's MODEL vs CCITT's MODEL

2.4 BASIC DEFINITIONS AND TERMINOLOGY

1. Communication Protocols :

Communication Protocols can be defined as the set of rules governing the orderly exchange of data and messages between peer communicating entities. Thus there would be a protocol at every level or layer of the ISO model described in the previous section.

2. Service Specifications :

Layer N provides a set of services to the user Layer N+1 above it by executing Layer N protocol, which is a set of functions, using the services provided by the Layer N-1 below it. The set of services provided by the Layer N is described by the Layer N Service Specifications. Layer N+1 service is a combination of Layer N service and Layer N+1 protocol.

Service specifications for a layer can be expressed in a Formal Description Language, [14, 17, 29]. Executable Service specifications in the form of a prototype can be obtained using a programming language. The programming language should be used with extreme care and in a strictly structured way so that no errors or ambiguities are introduced in constructing the prototype. This is the method we adopted for developing prototypes for Transport Service specifications and Class-0 Transport Protocol specifications, (see Chapter 3).

3. Protocol Specifications :

The Layer N Protocol Specifications describe how the Layer N Service is provided through specific sequences of iteractions between peer Layer N entities. The peer Layer N entities actually communicate using the Service provided by Layer N-1.

4. Service Primitives : Layer N Service Primitives describe the operations at the interface (access point) between the Layer N and Layer N+1 through which the Layer N Service is provided. Layer N+1 uses Layer N primitives for communication with the Layer N entity.

2.5 PROTOCOL SPECIFICATION ISSUES

Specification techniques developed for conventional software cannot be readily applied to specifying communication protocols because protocols have some time related properties [9, 10]. Protocols are implemented in software but their functionality involves a high degree of concurrency [8, 10]. It is important to note that imprecise service specifications of a layer, say layer n, can affect the services offered to layer n+1. For example if the specifications of the Transport Service are ambiguous, the Transport Layer would not be guaranteed to supply the set of services expected by the Session Layer. Therefore, to promote completeness and consistency, the service specifications of any layer should be laid down in an extremely precise and formal manner.

Formalizing requirements is the first step towards designing any piece of software. Imprecise definitions of protocol requirements give rise to imprecise, ambiguous and/or incorrect protocol specifications which result in faulty design.

Protocol errors can result from ambiguous and imprecise protocol specifications e.g Virtual Circuit establishment and Clearing in X.25 Packet level protocol as originally proposed by the CCITT, [5, 6].

Protocol Design errors can usually be attributed to the misinterpretation of the specification by the designer. Most protocol errors have been linked to erroneous design. Ambiguity in X.25 Virtual Circuit establishment state diagrams was a significant design error, [5, 6].

Protocol implementation errors arise due to incorrect programming, misinterpretation of correct design by the programmer etc. Most persistent errors found in protocols are specification and design errors.

2.6 EXAMPLES OF PROTOCOL DESIGN OR SPECIFICATION ERRORS

These examples have been put under a separate heading because the author believes that it sometimes becomes very difficult to attribute an error strictly to incorrect specifications or erroneous design. It should be noted here that the common protocol specification and design errors are usually categorized and listed as the following classes :

1. Deadlock condition

   This situation arises when two communicating entities wait indefinitely for the other either to complete or initiate an action. In the case of a deadlock, the entities are not exchanging any information. They, typically do not consume the networking resources such as CPU time.

   A number of deadlock conditions have been identified in the literature [4].

2. Infinite Message Looping

An Infinite Message Looping condition occurs when two communicating entities are in a loop such that, upon receipt of a message from one entity, the other responds with a message which in turn causes the first to repeat its first message transmission. This interchange continues indefinitely[4].

The effect on the communicating entities is similar to that of deadlocks i.e. further effective communication between the two entities is halted. However in an Infinite Message Looping condition, the entities are consuming the network resources and are therefore, more costly than deadlock conditions.

Discussion of the causes and effects of these conditions is beyond the scope of this thesis. The interested reader is referred to [4].

2.7 CURRENT PROTOCOL VALIDATION TECHNIQUES

For the sake of precision and accuracy, there is a need to express the specifications of a protocol in a Formal Description Language. These specifications can then be expressed in the form of executable specifications and can be tested for any errors or anomalies using protocol testing techniques. This section contains a survey of current Protocol validation techniques. The terms Validation and Verification are interpreted as being distinct in the context of

Software Testing technology. In this thesis the terms Protocol Validation and Protocol Verification both denote activities to ascertain that a protocol performs its function according to the set of original requirements. This activity could be the modelling and testing of the protocol by running a series of selected test cases through it and verifying it's correctness. Performance testing issues are not discussed here.

2.7.1 Protocol validation vs. Program validation

Protocol Validation techniques are quite similar to those used for Progam validation, [8, 10]. Some program validation techniques have been applied successfully to protocol validation [8,10]. To handle typical properties of protocols like concurrency, new approaches have been developed for protocol description e.g Finite State Automaton approach, Formal Description Techniques (FDT), [22, 23, 24, 29].

A formal classification of available techniques for Protocol validation is presented below. It should be noted at this point that this classification would be from the point of view of validation of protocols and not computer programs in general. Thus, the properties of protocols have been strictly kept in mind before attempting to make this classification. The major classes of protocol validation techniques are listed below with their explaination in the following subsections :

1. Reachability Analysis based techniques.
2. Formal Proving based techniques.
3. Symbolic Execution based techniques.

2.7.2 Reachability Analysis Based Techniques

Reachability analysis is based on exhaustively exploring all the possible interactions of two (or more) entities within a layer [17]. A composite state of the system is defined as a combination of the states of the cooperating protocol entities and the lower layer connecting them. From an initial state, all possible transitions are generated leading the system model to new system states. This process is repeated for each of the newly generated states until no new states are generated. The new system states are then analysed by the designer for error. For example, if no state transition is taking place, then there is a possible deadlock situation or a termination condition. Also, if the pattern of state transition is in the form of an infinite loop, then there could be a Ping-ponging situation.

Reachability Analysis is basically a Finite State Description based validation technique. It therefore requires a finite state model description of the protocol. Following are the types of finite state models which have been used to describe a protocol :

1. Duologue Matrix Model

The protocol is modelled as a pair of interacting processes, These processes are themselves represented as directed graphs [12]. A message transmission (represented by an integer) is represented by a pair of corresponding arcs, one in each graph. The process which sends the message has a negative value of the integer along the side of its graph while the process receiving the message has a positive value of the integer along the side of its graph. A sequence of transitions within a process is called a UNILOGUE and is represented by a path within its graph. A sequence of transitions between the two processes is called a DUOLOGUE and is represented by the pair of participating unilogues and their paths. A Duologue Matrix expressed as D = dij represents all the Duologues where dij is the Duologue formed by coupling the ith unilogue of the first process with the jth unilogue in the second process.

These Duologues can then be checked for particular kinds of erroneous behavior.

Limitations of the Duologue Matrix Model are :

a) Only two processes can be considered for analysis.
b) The number of duologues increases rapidly with increasing complexity of the state diagrams being validated. Therefore, Duologue Matrix analysis is

limited to the protocols which do not contain any cycles and therefore, return to their initial state after a finite number of interactions.

2. Perturbation Model

This is an extension of the Duologue Matrix model developed by C.H.West [18]. It overcomes one of the limitations of the Duologue Model i.e it does not require the interacting processes to come back to their initial states with the same periodicity. On the other hand, it covers all possible state transitions (starting from an initial state) by perturbing the current state of one of its component processes only (i.e all keeping all other components unchanged). Thus, new states of the whole system are achieved. This model handles many more protocols than the Duologue Model.

Example applications include CCITT Recommendations X.21 and the X.25 Packet level protocols [10].

3. Pure Finite State Model

This is a relatively primitive approach [10]. The communicating processes and the communication medium are represented by a Finite State Machine which as a whole represents the system. Transition from one state to another are triggered by internal or exter-

nal events. Validation is done on Reachability Analysis concept. This however, leads into the traversal of a massive reachability tree in case of complex protocols.

It has been used in validating the Call Set-up and Clearing procedure in X.25, [5, 10].

4. General Transition Model

This model combines the finite state machine approach and a programming language approach. Each communicating process is described in terms of

a) A finite state diagram.

b) A set of program variables.

A state transition is described by

c) A token indicating the transition's current position in the transition diagram.

d) The transition's effect on the system represented by a set of values of the program variables.

This combined approach allows complex models to be validated using the Reachability analysis technique and allows for software testing concepts to be used also.

It has been applied to ascertain the validity of X.25 Packet level protocol and HDLC protocol, [10].

2.7.3 Proving based techniques

A protocol, when considered as a program, can be subjected to program proving techniques [13]. For this purpose, the specifications of the protocol must be written in a very precise manner. Some of the program proving based techniques applicable to protocol validation are given below :

1. Inductive Assertion Method

   This technique of proving a protocol involves the attaching of statements called assertions at certain points of the program (protocol) which are required to be true. These could be values of boolean variables. Ideally, the assertions would be derived from the service specifications but since the definition of the services are not precise enough, the assertions could be formulated by the tester. The truth of an assertion follows from the truth value of the assertions at all points previous to it in the program hence the name Inductive Assertions [13]. After formulation of the assertions, it can be proved that the programs representing the protocol entity satisfy these assertions.

   The difference between program and protocol proving using this method is in the complexity of the model. Both programs and protocols can be modelled in terms of processes.

The completeness of a proof of correctness relies on the completeness of modelling of the processes and the communication medium, in having exercised all possible paths and verified every assertion [13].

2. AFFIRM AND SPEX

There is a specification language called SPEX used to write formal specifications of programs. This has been tailored to write protocol specifications [14]. There also exists a semi-automated verification system called AFFIRM which can be used to verify the correctness of an algorithm to realize a particular function in a computer network. An example of the application of this system to the CONNECTION ESTABLISHMENT function can be found in [14]. Furthermore, a connection protocol's specifications are expressed in SPEX in the same reference.

3. GYPSY

This is a software engineering system which provides a unified language for expressing both specifications and programs so that high level specifications can be progressively refined into detailed programs. Program modules can be both verified in advance or checked against their specifications at run time for particular inputs, [17].

2.7.4 Symbolic Execution based techniques

This approach to protocol verification is to some extent unique. Symbolic Execution is an effective program verification techniques in which the subject program is executed symbolically, i.e the variables take on symbolic values. Symbolic values are algebraic identifiers or expressions containing symbolic as well as numeric values, [8, 27].

For the purpose of protocol verification, this technique has been related to the Perturbation method of modelling protocols [8]. This brings it essentially under the heading of Reachability Analysis based techniques. The advantages of this technique when applied to the Perturbation model are as follows :

1. The protocol is interpreted by the verifier in essentially the same way as in reality. Therefore, it is easier to describe and verify protocols involving timing assumptions, counters and behavior affected by the actual contents of the message rather than merely by its arrival, [8].
2. Use of this technique substantially reduces the semantic size of the proof tree since symbolic rather than actual values are used and they prevent the global state space to explode.
3. By using the same technique for program and protocol verification, we can verify systems comprising both kinds of specifications.

A system which verifies protocols using Symbolic Execution (in relation with Perturbation model) is described in [8, 30]. It has been implemented as part of an existing program verification system called MCS (Microprogram Certification System), [30].

2.8 SOFTWARE TESTING TECHNIQUES AND RELATION TO PROTOCOL TESTING

There exists a diversity of Software Testing and Validation techniques which can be affectively tailored to the need of Protocol Validation.

Derivation of effective test sequences for protocol testing can be as difficult as for program testing. It is possible to enhance the existing specification based (Black Box) test case design techniques by adapting some existing testing strategies such as Functional Testing, Boundary Value Analysis and Error-Based Testing to meet the peculiar properties of communications protocols, [20]. It should be noted here that the enhancements to the existing methods should not be protocol dependent.

2.8.1 Protocol Validation -- Software Tools Perspective

This concept should be applied during the protocol design and implementation. There exist software tools which provide a mechanism for continuously monitoring the development of software. An example is CEMS (Continuous Evaluating and Mo-

nitoring System) which provides a number of interfaces e.g between the user/designer, designer/ tester etc, [31]. This or any such system could be extremely useful in coming up with a better design with fewer modifications to be carried out.

2.8.2 Automated Validation of Protocols

Automated validation of protocols has been carried out using Duologue Matrix Analysis, [12], (see section 2.7.2). A reformulated version of this theory was programmed in a system that can identify protocol errors automatically,, [7]. The underlying concept is that state diagram representation of a protocol facilitates the design of a system to automatically analyze its behavior. This validation process does not establish the correctness of semantics of the interaction i.e. whether or not the executable interactions accomplish a meaningful exchange of data [7].

This process was used for validation of the Call Establishment procedure of CCITT Recommendation X.21, to demonstrate its applicability in a real environment[7].

Chapter 3

PASCAL PROTOTYPES FOR TS AND CLASS-0 TP SPECIFICATIONS

This chapter presents Transport Service (TS) Specifications, Class-0 Transport Protocol (TP) Specifications and corresponding Pascal implementations in the form of prototypes. The TS Specifications described in section 3.1 are a combination of a finite state automaton model and an abstract data type model [22]. Its Pascal implementation in the form of a prototype is given in section 3.2. The Class-0 TP specifications are presented in the form of a state table in section 3.3, [24]. Its Pascal implementation in the form of a prototype is described in section 3.4. Experience gained by developing the prototype for TS specifications is utilized in modelling the Network Service Provider underlying the prototypes of the Transport Protocol Entities. Section 3.5 summarizes software design considerations and quality assurance activities.

3.1 TRANSPORT SERVICE SPECIFICATIONS

A brief description of the TS specifications is given in this section. This description is based on [22, 23] and is purposely kept brief. Detailed description can be found in [22, 23]. The TS specifications described below assume only a single transport connection between two users [22]. In practice, multiple connections would be permitted.

The functions in a transport layer are those necessary to bridge the gap between the services available from the network layer and those to be offered to the transport service users (such as a session layer). The set of transport services provided by the transport service provider (TSP) to its user consists of the services provided by all layers below it and the functions performed by the transport layer (see Fig.3.1 and Fig.3.7).

Consider the model of a Transport Connection given in Fig.3.1. There is an interaction of four entities namely two finite state Transducers and two queues. User A communicates with the transducer A through its TSAP. Transducer A communicates with its peer transducer B according to a set of specifications given in the state diagram of Fig.3.2. Transducer B on the other end recieves the incoming messages from its peer transducer, interprets them and gives an indication to User B through its TSAP according to the set of specifications given in Fig.3.2. User B initiates communica-

tion with User A in a similar fashion (this will be illustrated by an example in this section). The set of specifications of Fig.3.2 is for both transducers (considered peer entities). For the sake of concentrating on the TS specifications alone, the services provided by the lower layers are assumed to be complete and correct. We model the communication medium for data exchange as two queues, one per direction.

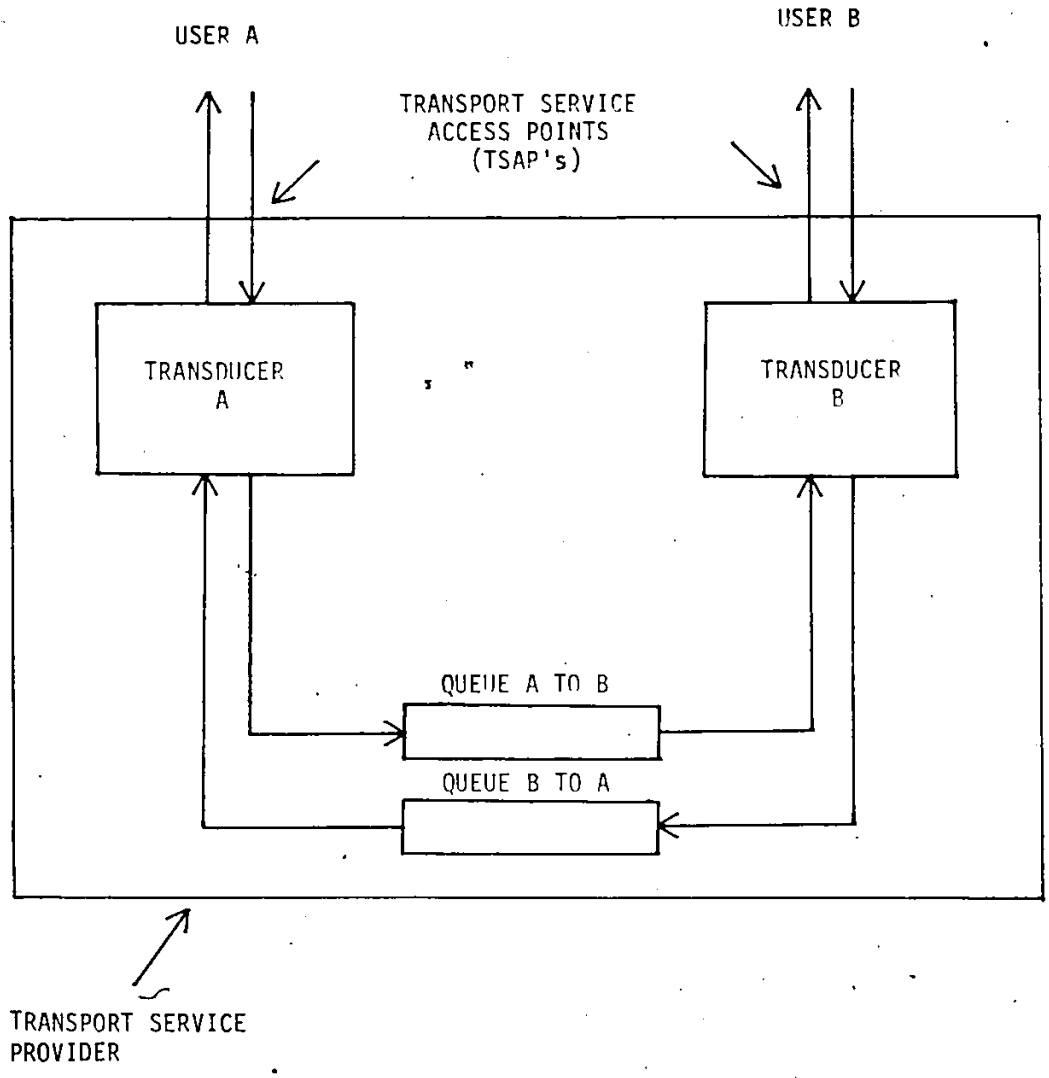


FIG. 3.1

MODEL OF A TRANSPORT CONNECTION

3.1.1 Transport Service Primitives.

The User interacts with its Transducer by means of Transport Service Primitives given below. It is to be noted that the TS primitives given below do not have service parameters for the implementation of the prototype of TS specifications given in section 3.2.

Transport Service Primitive	Description
TCREQ	Transport Connect Request. User sends to the Transducer.
TCIND	Transport Connect Indication. Transducer sends to the User.
TCRES	Transport Connect Response. User sends to Transducer.
TCCON	Transport Connect Confirm. Transducer sends to the User.
TDREQ	Transport Disconnect Request. User sends to Transducer.
TDIND	Transport Disconnect Indication. Transducer sends to the User.
TNODR	Transport Normal Data Request. User sends to Transducer.
TNODI	Transport Normal Data Indication. Transducer sends to the User.
TEXDR	Transport Expedited Data Request. User sends to the Transducer.
TEXDI	Transport Expedited Data Indication. Transducer sends to the User.

A single letter queue item is associated with each Primitive described above [22]. A table showing the input primitive to the Transducer, the corresponding queue item and an output primitive that the transducer generates after receiving that item from its peer Transducer is shown below.

Input Primitive	Queue Item generated / recognized	Output Primitive
TCREQ	q	TCIND
TCRES	s	TCCON
TDREQ	d	TDIND
TNODR	i	TNODI
TEXDR	x	TEXDI

3.1.2 Description of the Finite State Transducer.

According to the state diagram of Fig.3.2 based on [22] and rewritten according to our convention, the transducer can have four states, namely :

1. IDLE (1) : In this state the Transducer can :
   a) receive a TCREQ from its User, write a queue item 'q' on the queue and change its state to Out-Connect Pending state or
   b) receive a queue item 'q' from its peer, give a TCIND to its User and go to In-Connect Pending state.

In this state the transducer is waiting either for its own user to initiate a Transport connection establishment procedure or to respond to the initiation of a Transport connection from the user of its peer transducer.

2. OUT-CONNECT PENDING (2) : In this state, the Transducer can :

   a) receive a queue item 's' from its peer, give a TCCON to its User and go to Data Transfer Ready state or

   b) receive a TDREQ from its User, write a queue item 'd' on the queue and go to Idle state or

   c) receive a queue item 'd' from its peer, give a TDIND to its user and go to Idle state.

   In this state, the transducer knows that its own user initiated the Transport connection establishment i.e the TCREQ was Out-Going. It waits for a queue item 's' or a queue item 'd' from its peer transducer or a TDREQ from its own user.

3. IN-CONNECT PENDING (3) : In this state, the Transducer can :

   a) receive a TCRES from its User and go to Data Transfer Ready state or

   b) receive a TDREQ from its User, write a 'd' on the queue and go to IDLE state or

c) receive a queue item 'd' from the queue , write a TDIND to its User and go to IDLE state

In this state, the transducer knows that the user of its peer transducer initiated the Transport connection procedure i.e the TCREQ was In-Coming and is waiting for a TCRES or a TDREQ from its own user or a queue item 'd' from its peer transducer.

4. DATA TRANSFER READY (4) : In this state, the Transducer can :

a) receive a TNODR from its User, write a queue item 'i' on the queue and stay in Data Transfer Ready state or

b) receive a queue item 'i' from its peer, give a TNODI to its User and stay in Data Transfer Ready state or

c) receive a TEXDR primitive from its User, write a queue item 'x' on the queue and stay in Data Transfer Ready state or

d) receive a queue item 'x' from its peer, give a TEXDI to its User and stay in Data Transfer Ready state or

e) receive a TDREQ from its User, write a queue item 'd' on the queue and go to Idle state or

f) receive a queue item 'd' from its peer, give a TDIND and go to Idle state.

In this state, the transducer can process all kinds of user data requests and TDREQ from its user and the corresponding responses from the user of its peer transducer including a TDREQ which it receives in the form of a queue item 'd'.

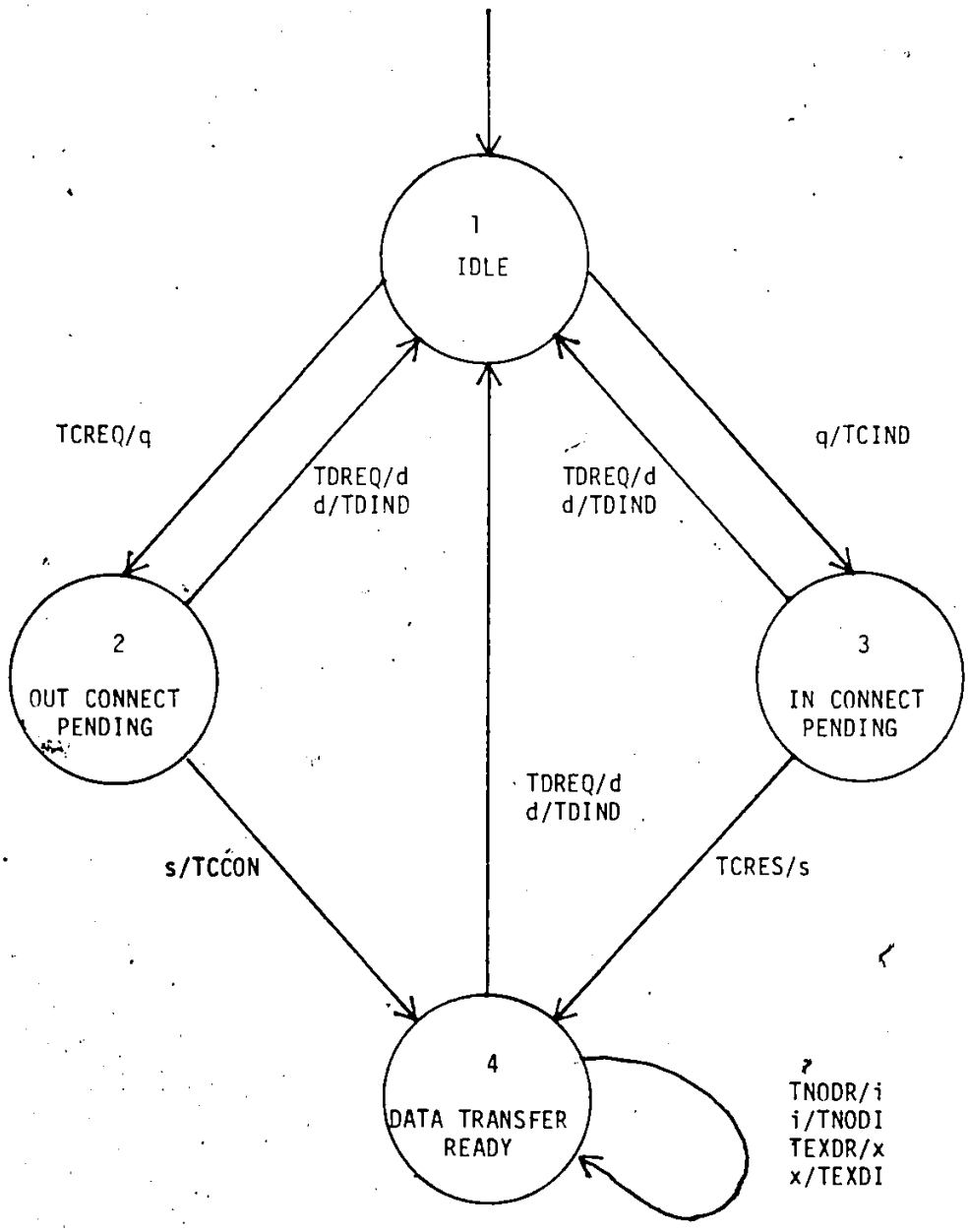


FIG. 3.2

STATE TRANSITION DIAGRAM (TRANSDUCER)

3.1.3 Queue Mechanism

There is one queue per direction as shown in Fig.3.1. A queue represents a flow control mechanism [22]. The two queues are created by the connection establishment procedure and destroyed by the connection release procedure. Details of the adding and removing of queue items to and from the queue can be found in [22].

The specifications of the queue mechanism requires Transport Expedited Data to have priority over all other queue items with the exception of the queue item 'd' associated with TDREQ, [22], but it will be shown in section 3.2 that in the prototype of TS specifications, the queues are actually under the control of the operating system and not the user. This implementation therefore, has certain restrictions which will be listed in the next section. The introduction of some limitations during implementation of specifications as a prototype is a typical phenomenon, [19]. Implementation limitations do not invalidate the usefulness of the TS specifications because this implementation is intended to test the functional behaviour of TS specifications in the form of a prototype. Performance behaviour is not under test in this study.

3.1.4 Transport Connection Establishment.....An Example

Consider the state diagram of Fig.3.2 and the Transport Connection model of Fig.3.1. Assume that both Transducer A and Transducer B are in IDLE state. The following sequence of operations takes place if User A wishes to establish a Transport Connection with User B. User B could also be the initiator of the Connection Establishment procedure with a similar sequence of operations but interchanging all A's and B's.

1. User A types in the primitive name TCREQ.
2. Transducer A reads in User A's request, writes the queue item 'q' on the queue A to B and goes to OUT CONNECT PENDING state.
3. Transducer B receives this queue item 'q' from queue A to B, gives a TCIND to User B and goes to IN CONNECT PENDING state.
4. User B types in the primitive name TCRES showing its willingness to accept the Transport Connection Request from User A.
5. Transducer B reads in User B's primitive TCRES, accordingly writes the queue item 's' on the queue B to A and goes to DATA TRANSFER READY state.
6. Transducer A reads in the queue item 's' from queue B to A, gives a TCCON (Transport Connect Confirm) primitive to User A, and goes to DATA TRANSFER READY state.

This is a simple, totally synchronized example of successful transport connection establishment. Actually, User A or User B have a choice of refusing to establish the connection at any intermediate stage during the connection establishment procedure.

More complex examples of Connection Establishment and Release shall be presented in Section 3.2.

3.2 A PASCAL PROTOTYPE FOR TS SPECIFICATIONS

This section presents the system used for the implementation, the design assumptions and implementation restrictions, the operating system directives used, a structural presentation of the various tasks and their communication, the queue mechanism implementation, task synchronization problems and a high level structural view of the task modules.

The Pascal routines for User A's side are identical to those for User B's side. Therefore, we present a description of these routines and their Pascal code for User A's side only. This would help in keeping the volume of the code small. Description of the routines for the prototype of TS specifications is given in Appendix A. Pascal code for these routines is given in Appendix B.

3.2.1 System Used

The TS specifications described in the section 3.1 were implemented in the form of "executable specifications" using the following system :

1. The computer used was PDP 11/34 with Digital VT100 and Lanparscope XT100 type terminals.
2. The operating system used was RSX-11M Version 3.2. This operating system was chosen because it provides certain system directives such as SEND DATA and RECEIVE DATA for intertask communication. Such a communication capablity between independent tasks is critical to the implementation of the TS specifications. Explanation and examples of these and other system directives can be found throughout this section and in section 3.4. Details can be found in [32].
3. The programming language used was Swedish Pascal. This version of Pascal was used because :
   a) It permits use of external procedures,
   b) It allows access to external Fortran routines,
   c) It allows a graceful exit from the program using the HALT primitive in case of an error.

Some comments on the portability of the code are presented in section 3.5.

Pascal provides constructs such as the CASE statement which can be used to represent a state table in a procedural programming language. The label of a CASE statement can be used to represent the different states while the action to be taken in that state is represented by the logic in front of the label, e.g.

```
CASE state OF

one  :  begin
          receive              (* user request *)
            .
            .                  (* other statements *)
            .
        end;

two  :  begin
          receive              (* a queue item *)
            .
            .
            .
        end;
  .
  .
  .
end;                           (* CASE *)
```

Therefore by looking at the code, the state diagram representation of the specifications can be readily compared

to the code. Each label shown as a state in the above example represents a block of information related to that state only. State transitions can be followed by tracing the execution path through the Pascal program. Representation of TS specifications in Pascal using the CASE construct is close to the state diagram representation of the specifications but are executable.

3.2.2 Design Assumptions and Implementation Restrictions

Certain design assumptions and implementation restrictions were made for developing a prototype for the TS specifications. In this study the performance behaviour of the TS specifications is not under consideration since only a prototype has been developed. Emphasis is on functional behaviour of TS specifications. Therefore, the following design assumptions can be safely made without invalidating the usefulness of the TS specifications :

1. Queue A to B and Queue B to A of Fig.3.1 are implemented as system queues. The reason for this restriction is that the RSX-11M operating system directives SEND DATA requires the address of the receiving task and creates a queue for each task (to which data is to be sent). We therefore, use these system queues as the queues in the model of the Transport Connec-

tion (see Fig.3.1). The restriction therefore is that these queues are under the system's control and no queue item has priority over the others. The receiving task is automatically made aware of the origin of the incoming data. As an extension of this design, the queues shown in Fig.3.1 can be made independent tasks which can give priority to any particular kind of data on the queues over the others.

2. The global event flags used to synchronize the tasks of the implementation of TS specifications as a prototype must be chosen such that they are not used by the operating system for its own purposes
3. The TS specifications implemented here are assumed to support a single Transport connection between two user tasks. There is an implicit convention in the TS specifications that each TCREQ initiates a unique Transport connection. This assumption does not restrict the testing of TS specifications supporting only one transport connection at a time. No capability is provided to support multiple connections between two users.
4. Due to the preceding assumption, there is a possibility of a deadlock situation in the event of both User A and User B attempting to request a Transport connection by sending a TCREQ at the same time (i.e both users issue a TCREQ before anyone of them is aware of

a connection establishment initiation attempted by the other). This would result in both transducers going into OUT-CONNECT PENDING state. To prevent this, a deadlock prevention scheme has been proposed and implemented. This is described in section 3.2.6.

5. We have introduced a query command called QSTAT which either user can give as a user request and his respective transducer displays the current state the transducer is in. This command is useful because it gives the user a chance to check the state of his transducer.

3.2.3 RSX-11M System Directives used.

Following are the major system directives used in our implementation of TS specifications, [32] :

1. SEND DATA

This directive is actually a Fortran system routine which can also be invoked from Pascal programs. It instructs the system to queue (FIFO) a 26 byte (13-Word) block of data for the destination task. When a local event flag is specified, the indicated event flag is set for the SENDING task. Declaration of the directive in the calling Pascal routine is as follows :

```
Procedure send (var tsk : integer;
                var buf : buff;
```

```
                    var efn : integer;
                    var ids : integer) extern(fortran);
```

where :

a) tsk = Destination Task name.

b) buf = 26-byte block of data to be sent.

c) efn = Local event flag number.

d) ids = Directive status.

2. RECEIVE DATA

This directive requests the system to dequeue a 26 byte (13-word) block of data for the issuing task. This data block has been queued (FIFO) for the task via a SEND DATA directive. The 26-byte data block and the sender task's name are returned in an indicated 30-byte buffer, with the sender task name in the first two words.

Declaration of this directive in the calling Pascal routine is as follows :

```
Procedure receiv (var tsk : integer;
                  var buf : buff;
                  var ids : integer);extern(fortran);
```

where :

a) tsk = Sender task name.

b) buf = 30-byte data block of received data.

c) ids = Directive status.

It should be noted here that SEND DATA and RECEIVE DATA directives will eventually be used to send and receive control primitives of the TP specifications while actual data exchange will be done using two other system directives, SEND BY REFERENCE and RECEIVE BY REFERENCE. These directives will be described in section 3.4.

3. MARK TIME

The Mark Time directive instructs the system to declare a significant event after a given time interval. The interval begins when a task issues the directive. If an event flag is specified, the flag is cleared when the directive is issued and set when the significant event occurs.

Declaration of the directive in the calling Pascal routine is as follows :

```
procedure mark(var efn : integer;
               var tmg : integer;
               var tnt : integer;
               var ids : integer);extern(fortran);
```

where :

a) efn = Event flag number.

b) tmg = time interval magnitude.

c) tnt = time interval unit code (for seconds, minutes etc.).

d) ids = Directive status.

This directive is used along with the directive WAITFR to put a task to sleep for a specified amount of time. MARK instructs the system to set a specified event flag (efn) after a certain time interval. Immediately following the call to MARK directive the directive WAITFR is invoked. WAITFR directive instructs the system to stop the execution until the event flag specified in its parameter is set. Therefore, by giving the same value to the event flag in both directives (MARK and WAITFR), we can put the task to sleep. MARK would instruct the system to set the event flag after the specified time. Immediately following it, WAITFR would stop the execution until its event flag is set. After the time specified in MARK, the system sets the event flag and the task wakes up (i.e execution starts).

A combination of these two directives can be used effectively to stop execution of a task for a specified time interval. The system overhead can therefore, be reduced in case many tasks are running simultaneously.

In addition to the above mentioned directives, two more system directives called SETEF and CLREF are used. SETEF instructs the system to set a specified global event flag. CLREF instructs the system to clear a specified global event flag. Details can be found in [32].

Use of these system directives in our implementation was mainly due to D. Lefebvre as part of a fast protocol prototyping project directed by Dr. R. L Probert of University of Ottawa.

3.2.4 Task Structure and Communication

A structural diagram showing the interactions of various tasks involved in the implementation is presented in Fig.3.3. There are four tasks running independently :

1. TERMA : This task simply reads requests typed in by the User A and sends them to Transducer A.
2. TRNSDA : This task is the Transducer A of Fig.3.1 and runs according to the specifications given in the state diagram of Fig.3.2. TRNSDA can receive data from TERMA or from its peer Transducer, TRNSDB.
3. TERMB : This is the peer task of TERMA. TERMB is identical in form to TERMA and reads in user requests from User B.
4. TRNSDB : This is the peer Transducer of TRNSDA. TRNSDB is identical in form to TRNSDA.

A simple example of communication of the four tasks is given below in the form of a sequence of operations. Concerns related to synchronization of these tasks will be addressed later. This example is intended only to illustrate the basic mode of communication among these tasks. The example describes the Transport Connection establishment proce-

dure described earlier in terms of task communication. Assume that tasks TRNSDA and TRNSDB are initially in IDLE state.

1. User A types in a TCREQ on his terminal.
2. Task TERMA reads in that request at once and uses the SEND DATA directive to send it to task TRNSDA. The SEND DATA directive has task TRNSDB's address as one of its parameters. It therefore, puts the TCREQ in the task queue A (see Fig.3.3). This queue can be thought of as being logically attached to TRNSDA.
3. Task TRNSDA reads this TCREQ from the task queue A using the RECEIVE DATA directive of RSX-11M and sends the queue item 'q' to task TRNSDB via the SEND DATA directive. TRNSDA changes its state to OUT-CONNECT PENDING.
4. Task TRNSDB reads in this request using the directive RECEIVE DATA from its task queue B, writes a TCIND to User B and changes its state to IN-CONNECT PENDING.
5. User B then types in a TCRES which is read in by the Task TERMB which sends it to the task TRNSDB using the SEND DATA directive. This TCRES is written on the task queue B.
6. Task TRNSDB receives the TCRES, sends a queue item 's' to the Task TRNSDA and changes its state to DATA TRANSFER READY.

7. Task TRNSDA receives this queue item 's' from its task queue A, writes a TCCON to User A and changes its state to DATA TRANSFER READY state.
8. User A and User B can now exchange data using SEND DATA and RECEIVE DATA directives and through the task queues A and B.

Table 3.1 shows various other test interactions applied to the prototype of TS specifications. Test examples involving strict synchronization of various requests are presented in subsection 3.2.6.

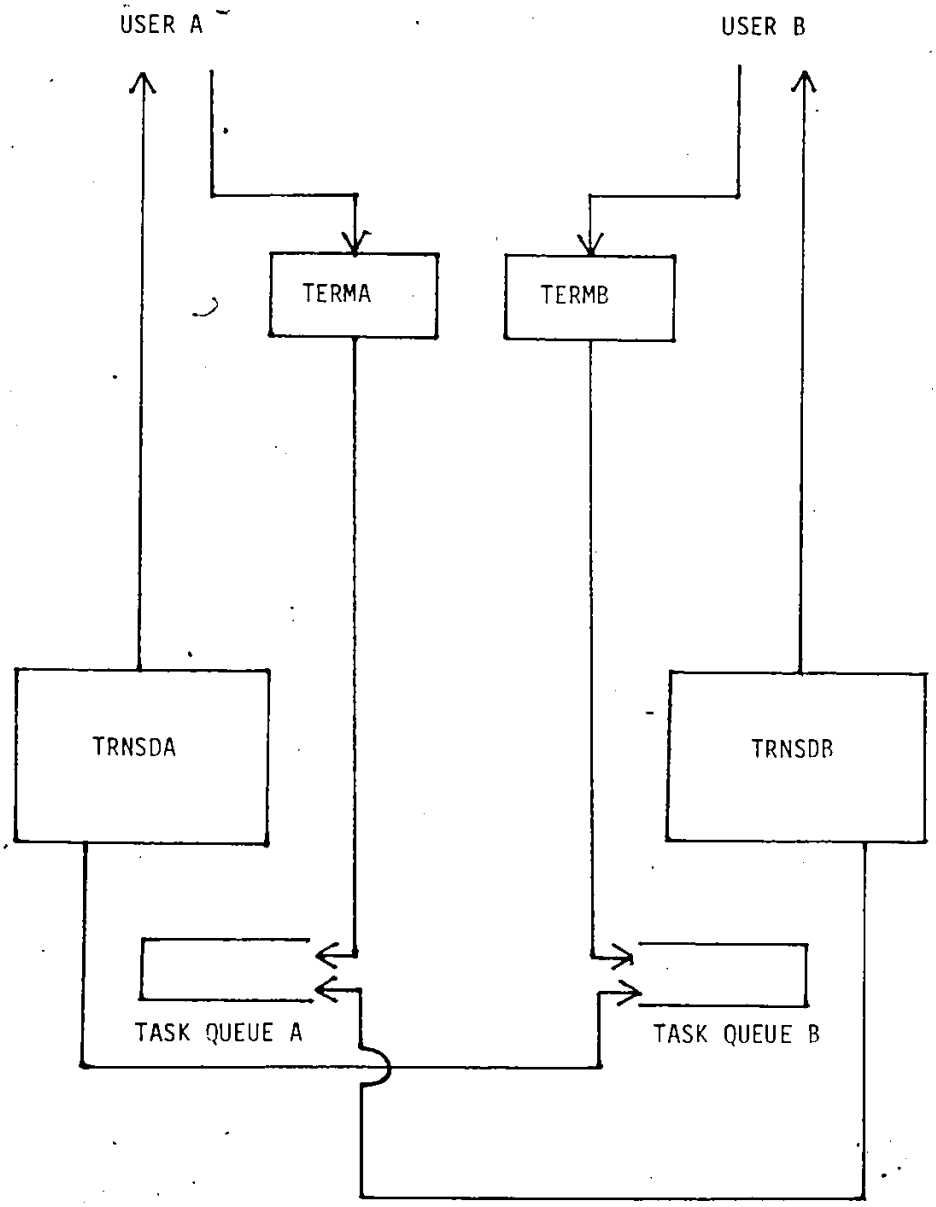


FIG. 3.3

INTERACTION OF TASKS
(TRANSPORT SERVICE SPECIFICATIONS IMPLEMENTATION SCHEME)

TEST CASE NO.	USER A INPUT	OUTPUT TO USER B	USER B INPUT	OUTPUT TO USER A
1.	TCREQ TDREQ QSTAT	TCIND TDIND TRNSDB IS NOW IN IDLE STATE	TCRES QSTAT	TCCON TRNSDA IS NOW IN IDLE STATE
2.	TCREQ QSTAT	TCIND TRNSDB IS NOW IN IN-CONNECT PENDING STATE	QSTAT TDREQ	TRNSDA IS NOW IN OUT-CONNECT PENDING STATE TDIND
3.	TCREQ TDREQ QSTAT	TCIND TDIND TRNSDB IS NOW IN IDLE STATE	QSTAT	TRNSDA IS NOW IN IDLE STATE
4.	TCRES QSTAT TNODR	TCCON TRNSDB IS NOW IN DATA TRANSFER READY STATE TNODI	TCREQ QSTAT TEXDR TDREQ	TCIND TRNSDA IS NOW IN DATA TRANSFER READY STATE TEXDI TDIND

TABLE 3.1

TEST EXAMPLES APPLIED TO EXECUTABLE TS SPECIFICATIONS

3.2.5 Queue Mechanism of Implemented system

The Task Structure of Fig.3.3 is similar to the Model of Transport Connection of Fig.3.1 except for the queue structure. The queues shown in Fig.3.3 are system queues and are completely controlled by the host system. The SEND DATA directive queues up the data items sent to any task in a system queue reserved for that task. Any data sent to this task (no matter what source) is queued in the same queue. Therefore, these system queues are shown in Fig.3.3 as being logically attached to the task that can receive data. If it is desired to test the executable TS specifications with arbitrary behaviour of the underlying communication medium, the queues shown in Fig.3.1 can be made independent tasks and data flow can be controlled by those tasks as an extension of this implementation.

3.2.6 Synchronization Considerations : An Example Problem

Due to the queue mechanism of our implementation, in the absence of a synchronization strategy among the four tasks, a deadlock could arise. For example, consider the case when both User A and User B initiate a Transport Connection Establishment procedure at the same time. Suppose that both transducers are currently in IDLE state (see Fig.3.1 and Fig.3.2). The following sequence of events will take place :

1. Both User A and User B type in a TCREQ at the same time.

2. Since there is no synchronization, both transducer A and transducer B send a queue item 'q' to each other and change their states to OUT-CONNECT PENDING (see Fig.3.2),
3. In OUT-CONNECT PENDING state, a transducer can :
   a) receive a queue item 's' or a queue item 'd' from its peer transducer or
   b) receive a TDREQ from its own user.
4. Now, since both transducers are in OUT-CONNECT PENDING state, neither can receive a TCRES from its user and change its state to DATA TRANSFER READY. Therefore, there is a deadlock.
5. The only thing possible in this situation is to initiate a Transport Disconnection procedure from either end as a result of time-out.

3.2.6.1 Deadlock Prevention Scheme

To prevent the above deadlock situation, a deadlock prevention scheme has been implemented. This scheme uses four global event flags to synchronize the four tasks. All tasks are active but they are constrained to receive data according to the time sequence diagram of Fig.3.4.

The scheme is such that control to receive/send is swapped from side A to side B and vice versa e.g. transducer A tries to receive an incoming request (from its user or from the peer transducer) for a predefined length of time (which

could also be zero seconds). If there is an incoming request, transducer A processes it according to the state diagram of Fig.3.2. If there is no incoming request from the user or the peer transducer, it sets a global event flag which allows transducer B to receive from its task queue. Transducer B repeats the same process. Therefore, the priority to receive is swapped between transducer A and transducer B. In addition to this, tasks TERMA and TERMB are also linked with their transducers. When TERMA has sent a user request to TRNSDA, it sets an event flag which allows TRNSDA to receive User A's request. TERMA then clears its own event flag. TRNSDA receives User A's request, processes it and after a certain predefined length of time, it sets an event flag for task TERMB signalling it to send User B's request (if any) to TRNSDB. TRNSDA then sets the event flag for TRNSDB to receive and clears its own event flag. TRNSDB repeats the same procedure. Therefore, even if the two users type in a request and hit RETURN at the same time, either task TERMA or TERMB would have its event flag set. Therefore, one would be able to send its user's request to its transducer before the other one.

Now, if a TCREQ is typed in by both users at the same time, only one of the transducers will be in a position to receive because of the continuous swapping of control to receive. The transducer in receiving position (say transducer A) receives TCREQ from its user first, writes a queue item

'q' on transducer B's task queue, changes its state to OUT-CONNECT PENDING and sets an event flag signalling transducer B to receive. 'Transducer B receives the queue item 'q', sends a TCIND to User B, changes its state to IN CONNECT PENDING and then receives the TCREQ send initially by User B. But, Transducer B has already changed its state to IN CONNECT PENDING, therefore, it cannot accept a TCREQ from User B in this state (see Fig.3.2) and treats it as an erroneous request giving an error message to its user.

Similarly, if the two users initiate a Transport Disconnection procedure at the same time, one of the two transducers will receive the TDREQ from its user first, send a queue item 'd' to its peer transducer and change its state to IDLE. The peer transducer receives the queue item 'd', gives a TDIND to its user and changes its state to IDLE. Now it receives the TDREQ sent by its user. Since it is in IDLE state, it treats the TDREQ as an erroneous request and gives an error message to its user. Use of global event flags restricts the two users to be on the same computer.

There are no guidelines in the TS specifications for the synchronization of the various tasks. Synchronization is also an issue related to the particular operating system which is being used for the implementation of the TS specifications.

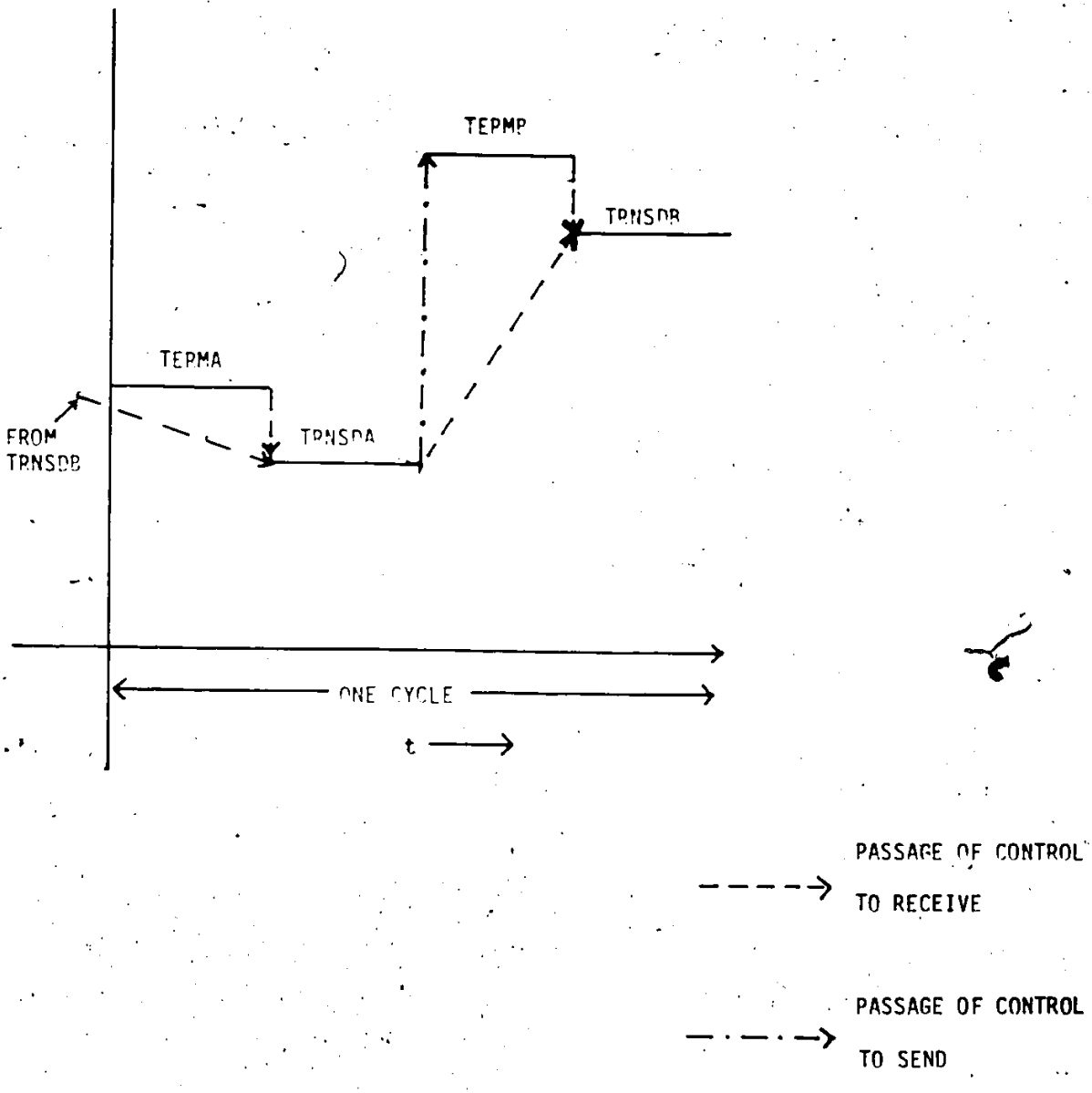


FIG. 3.4

DEADLOCK PPEVENTION SCHEME

3.2.7 Error Handling

The implemented system treats two types of erroneous requests :

1. Those sent by the user to its Transducer and
2. Those sent by a Transducer to its peer Transducer.

The first one is dealt by flashing an error message on the user's screen which shows the erroneous request and the current state of the Transducer. The second one is dealt by sending an error queue item 'e' to the transducer which sent the erroneous queue item and printing an error message on its users screen.

3.2.8 High Level Structure of Modules

A high level flow diagram for the task TERMA is presented in Fig.3.5 and a similar diagram for the task TRNSDA (Transducer A of Fig.3.3) in Fig.3.6. Since tasks TERMB and TRNSDB (Transducer B) are identical to tasks TERMA and TRNSDA respectively, their flow diagrams are ommitted.

These diagrams give a high level view of the various procedures system directives and functions involved and their relation to each other. The calling routine is the tail of the arrow while the called routine is the head of the arrow. All procedures and functions shown in Fig.3.5 and 3.6 are external to their calling modules. They have been coded and compiled in separate files and invoked from

the calling modules. This allows for easy modification and maintainability of the system.

The name, function, usage, input parameters, output parameters etc. of all routines shown in Fig.3.5 and Fig.3.6 can be found in Appendix A.

Software design considerations and software testing strategy adopted will be discussed in section 3.5.

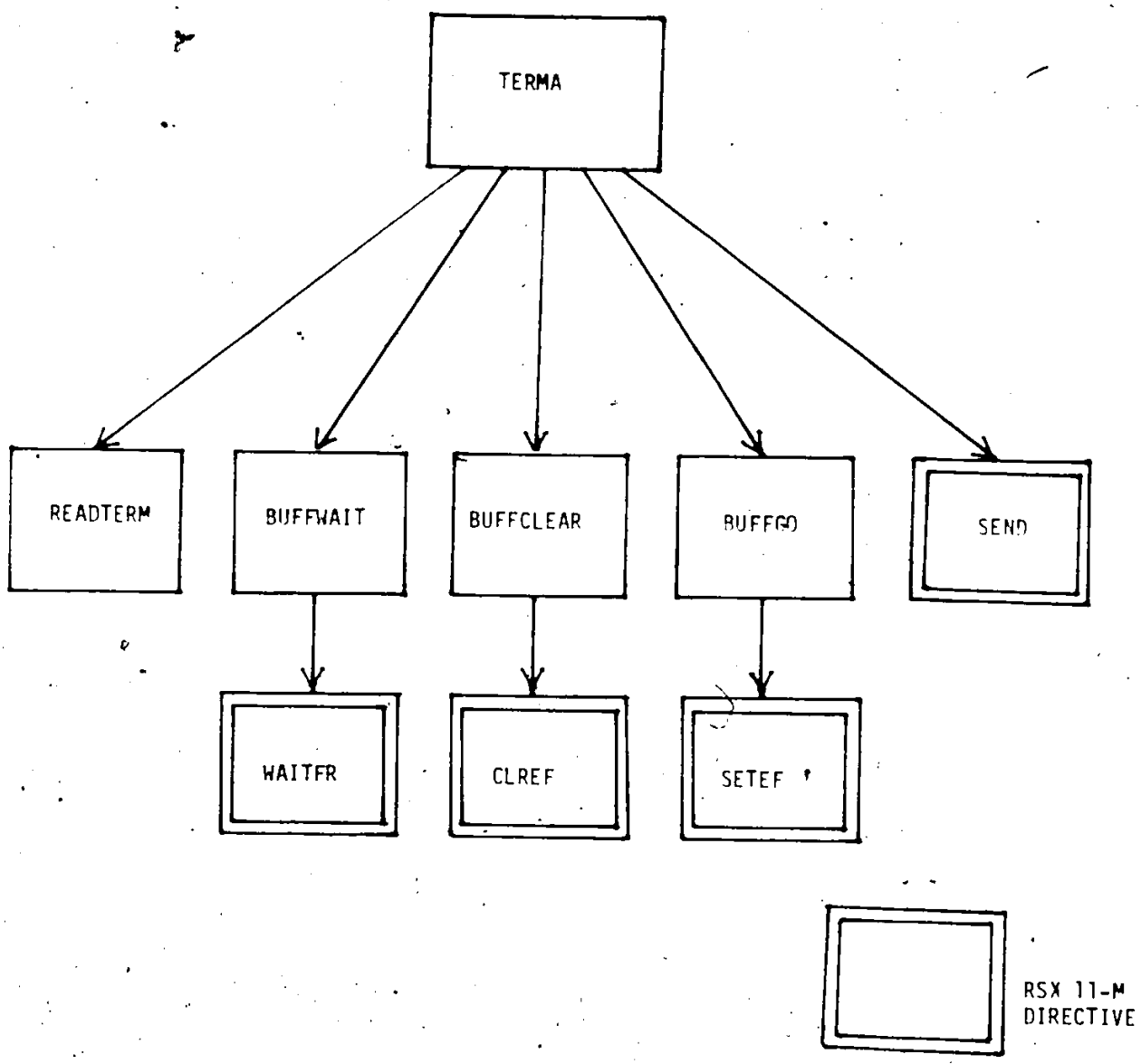


FIG.3.5

HIGH LEVEL STRUCTURE OF MODULES FOR TASK TERMA

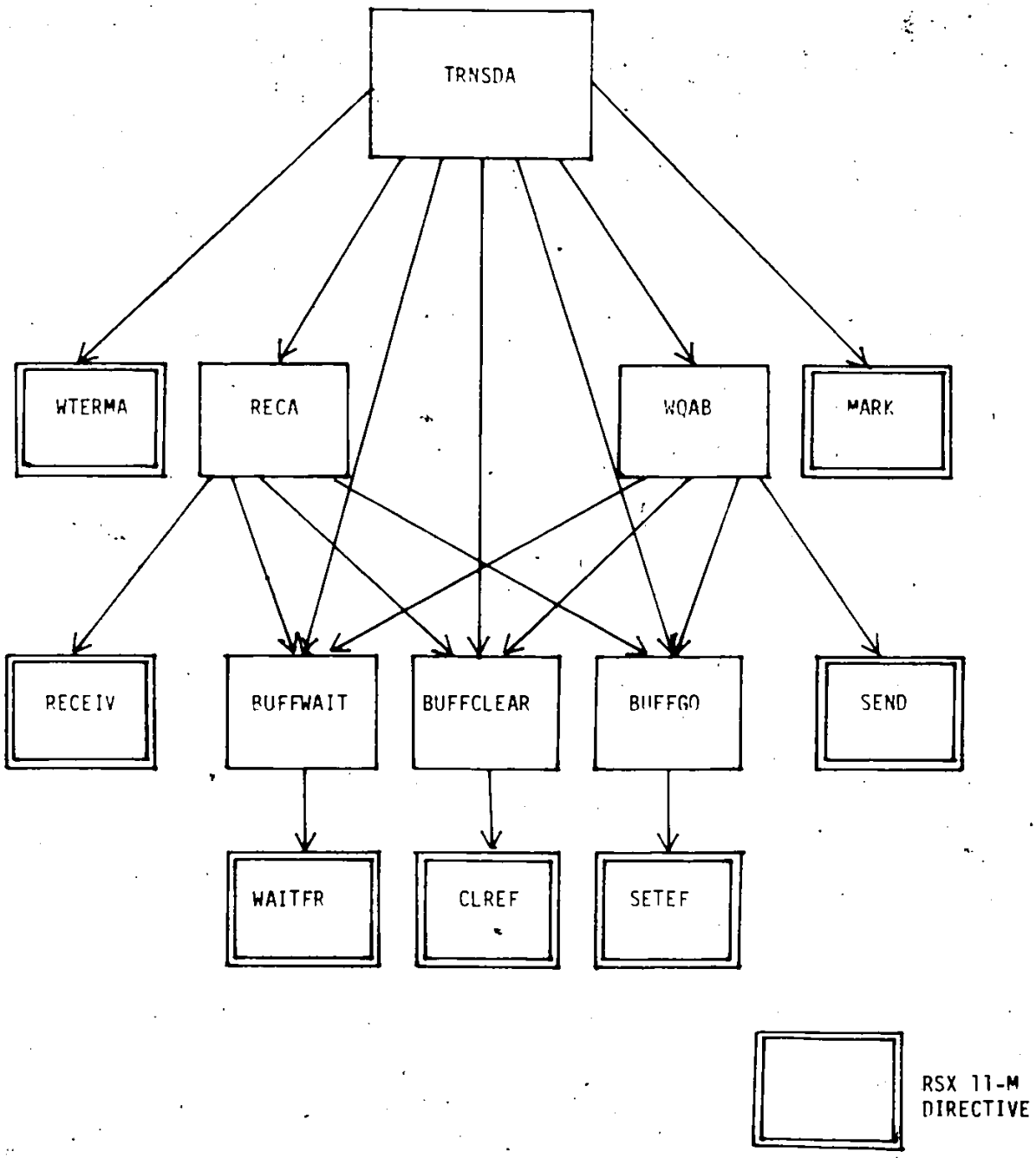


FIG. 3.6

HIGH LEVEL STRUCTURE OF MODULES OF TASK TRNSDA (TRANSDUCER A)

3.3 CLASS-0 TRANSPORT PROTOCOL SPECIFICATIONS

Class-0 Transport Protocol specifications presented below have been extracted from ISO document [24] which presents the specifications for all five classes of Transport Protocol. The Transport Service user interacts with the Transport Protocol Entity by Transport Service Primitives described in the previous section and the Protocol Entity interacts with the Network Service Provider below it by means of Network Service Primitives according to the state table described in Tables 3.2 (a), 3.2 (b), 3.2 (c) and 3.2 (d). An outline of the state table is given below.

3.3.1 State Table for Class-0 Transport Protocol Entity

The state table shown in Tables 3.2 (a) and 3.2 (b) is only for Class-0. This state table describes the various states the Transport Protocol Entity can take (see Fig.3.7), the action/s that should be taken in that state upon reception of an input and the resultant state, [24]. Predicates and specific actions to be taken are shown in Tables 3.2 (c) and 3.2 (d) respectively. Conventions used are as follows :

1. The Protocol Entity can have incoming and outgoing events e.g a TCREQ is an incoming event for the Protocol Entity from the Transport Service user above it and a CR (Connect Request) is an incoming event from the Network Service Provider. Similarly, there are outgoing events from the Protocol Entity to the

Transport Service user above it such as a TCCON and to the Network Service Provider such as an NDREQ (Network Disconnect Request). These events are shown in the first column of the state table in their abbreviated form. It is worthwhile to mention here that the Transport Service Primitives described in section 3.1, contain certain parameters such as the Source Address, the Destination Address, Quality of Service parameters etc. [23, 24]. A summary of these primitives and their parameters is given in Table 3.3. There also exist standard formats for various kinds of Transport Protocol Data Units (TPDU's) for example a CR TPDU is a Connect Request TPDU. A summary of these TPDU's is given in Table 3.4. Details of their format can be found in [24].

2. All states that the Transport Protocol Entity can assume are shown in the top row of the table in their abbreviated and expanded form. In OPEN state, the Transport Protocol Entity can exchange Transport User data with its peer entity. In CLOSED state the Transport Protocol Entity is dormant i.e it is not performing any function. The other states have self explanatory names.
3. Corresponding to each state-event pair is an action or set of several actions to be taken. In addition there are are conditional actions which are written as :

```
P1 : DR
     CLOSED;
NOT P1 :
          TCIND
          WFTRESP
```

This can be interpretted as :

```
if (P1) then
begin
  send a DR    (* Disconnect Request *)
  change state to CLOSED
end;

if (NOT P1) then
begin
  send a TCIND
  change state to WFTRESP
end
```

Invalid state-event combinations are left blank [24].

STATE / EVENT	WAIT FOR NETWORK CONNECTION (WFNC)	WAIT FOR CONNECT CONFIRM (WFCC)	OPEN	WAIT FOR TRANSPORT RESPONSE (WFTRESP)	CLOSED
TCREQ					P0:TDIND CLOSED; P2:NCREQ WFNC; P3:CR WFCC; P4:WFNC
TCRESP				CC OPEN	
TNODR			[3] OPEN		
TDREQ	[1] CLOSED	P5: NDREQ CLOSED	P5: NDREQ CLOSED	DR CLOSED	

TABLE 3.2 (a)

STATE TABLE FOR TRANSPORT PROTOCOL ENTITY ,[24]
(INCOMING EVENTS FROM TRANSPORT SERVICE USER)

STATE / EVENT	WAIT FOR NETWORK CONNECTION (WFNC)	WAIT FOR CONNECT CONFIRM (WFCC)	OPEN	WAIT FOR TRANSPORT RESPONSE (WFTRESP)	CLOSED
NCCON	CR WFCC				
NRIND	TDIND [1] [5] CLOSED	TDIND [1] [5] CLOSED	TDIND [1] [5] CLOSED	TDIND [1] [5] CLOSED	
NDIND	TDIND CLOSED	TDIND CLOSED	TDIND CLOSED	TDIND CLOSED	
CR TPDU					P1: DR CLOSED NOT P1:TCIND WFTRESP
DR TPDU		TDIND [1] CLOSED	P5: NDREQ CLOSED;		
CC TPDU		P8: TCCON OPEN 5 P6 AND P5: TDIND NDREQ CLOSED			DR CLOSED
DT TPDU			[3] OPEN		CLOSED
ER TPDU		TDIND [1] CLOSED	NDREQ [2] CLOSED		CLOSED
AK TPDU	ER CLOSED				
EA TPDU	ER CLOSED				
ED TPDU	ER CLOSED				
DC TPDU	ER CLOSED				

TABLE 3.2 (b)

STATE TABLE FOR TRANSPORT PROTOCOL ENTITY ,[24]
(INCOMING EVENTS FROM NETWORK ENTITY/PEER TRANSPORT PROTOCOL ENTITY)

NAME	DESCRIPTION
P0	T-CONNECT REQUEST UNACCEPTABLE
P1	UNACCEPTABLE CR TPDU
P2	NO NETWORK CONNECTION AVAILABLE
P3	NETWORK CONNECTION AVAILABLE AND OPEN
P4	NETWORK CONNECTION AVAILABLE AND OPEN IN PROGRESS
P5	CLASS IS CLASS-0 (CLASS SELECTED IN CC)
P6	UNACCEPTABLE CC
P8	ACCEPTABLE CC

TABLE 3.2 (c)

PREDICATES FOR CLASS-0 TP SPECIFICATIONS,[24]

NAME	DESCRIPTION
[1]	IF THE NETWORK CONNECTION IS NOT USED BY AN OTHER TRANSPORT CONNECTION ASSIGNED TO IT, IT MAY BE DISCONNECTED
[2]	RECEIPT OF AN ER TPDU (SEE SEC. 6.22 OF [24])
[3]	SEE DATA TRANSFER PROCEDURES OF THE CLASS
[4]	SEE EXPEDITED DATA TRANSFER PROCEDURE OF THE CLASS
[5]	AN N-RESET RESPONSE HAS TO ISSUED ONCE FOR THE NETWORK CONNECTION IF THE NETWORK CONNECTION HAS NOT BEEN RELEASED

TABLE 3.2 (d)

SPECIFIC ACTIONS FOR CLASS-0 TP SPECIFICATIONS ,[24]

3.3.2 Transport Connection Establishment.... An Example

This subsection presents an example of the Transport Connection Establishment procedure assuming :

1. There exist two protocol entities which change states and performs actions according to the state table of Tables 3.2 (a) and Table 3.2 (b) described above.
2. Every Transport Service Primitive and the TPDU's have certain parameters, (see Tables 3.3, 3.4 and 3.5), (see 24 for TPDU formats).

Consider the diagram of Fig.3.7. Suppose that both the Transport Protocol Entities are in CLOSED state, there is no Network connection available and User A wishes to establish a Transport Connection with User B. The following sequence of operations is appropriate :

1. User A types in TCREQ alongwith various parameters e.g the source address, the destination address and quality of service parameters[23, 24].
2. Transport Protocol Entity A receives this request, sends a NCREQ (Network Connect Request) to Network Entity A (to establish a Network Connection with its peer Network Entity B) and changes its state to WFNC.
3. The Network Entity A negotiates a Network Connection with its peer Network Entity B.
4. When this Network Connection is succesfully established, i.e both Network Entities are in Data Transfer Ready state, Network Entity A sends a NCCON (Net-

work Connection Confirm) to Transport Protocol Entity A.

5. Transport Protocol Entity A which is in WFNC state, receives the NCCON from Network Entity A, sends a CR TPDU which has all of the parameters of the TCREQ primitive sent by Transport Service User A to Network Entity A and changes its state to WFCC. This CR TPDU is sent to the Network Entity A in the User Data Field of a NNODR Network Service Primitive.
6. Network Entity A receives the NNODR primitive and sends the Network Normal Data queue item to its peer Network Entity with the CR TPDU as its parameter.
7. Network Entity B receives this queue item, seperates the CR TPDU from it and sends it to Transport Protocol Entity B as a NNODI Network Service Primitive. Transport Protocol Entity B is currently in CLOSED state.
8. Transport Protocol Entity B receives this CR TPDU, sends a TCIND to User B along with the information carried by the CR TPDU in its parameters and changes its state to WFTRESP.
9. User B now types in a TCRES primitive along with its parameters.
10. Transport Protocol Entity B receives this TCRES, sends a CC TPDU to Network Entity B in the User Data field of a NNODR Network Service Primitive (to be

sent to Transport Protocol Entity A) and changes its state to OPEN.

11. Network Entity B receives the CC TPDU from Transport Protocol Entity B, concatenates it to the User Data field of the Network Normal Data queue item and sends the queue item to Network Entity A.
12. Network Entity A receives this queue item, separates the CC TPDU from it and sends the CC TPDU to Transport Entity A as a parameter of the NNODI Network Service Primitive. Transport Protocol Entity A is currently in WFCC state.
13. Transport Protocol Entity A receives the CC TPDU, sends a TCCON to User A along with the information contained in the CC TPDU such as the responding address and the quality of service parameters and changes its state to OPEN.
14. Both Transport Protocol Entities are now in OPEN state i.e they have successfully established a Transport Connection and can now exchange User Data.

User B can also initiate the Connection Establishment procedure with the same sequence of operations but in the opposite direction. It should be noted here that the above example is a simplified version of Transport Connection Establishment in that both User A and User B remain willing to establish the Transport Connection during the whole process of connection establishment. In reality, either of the two

users may initiate a Transport Disconnect procedure by sending a TDREQ primitive at any stage of Transport Connection Establishment procedure.

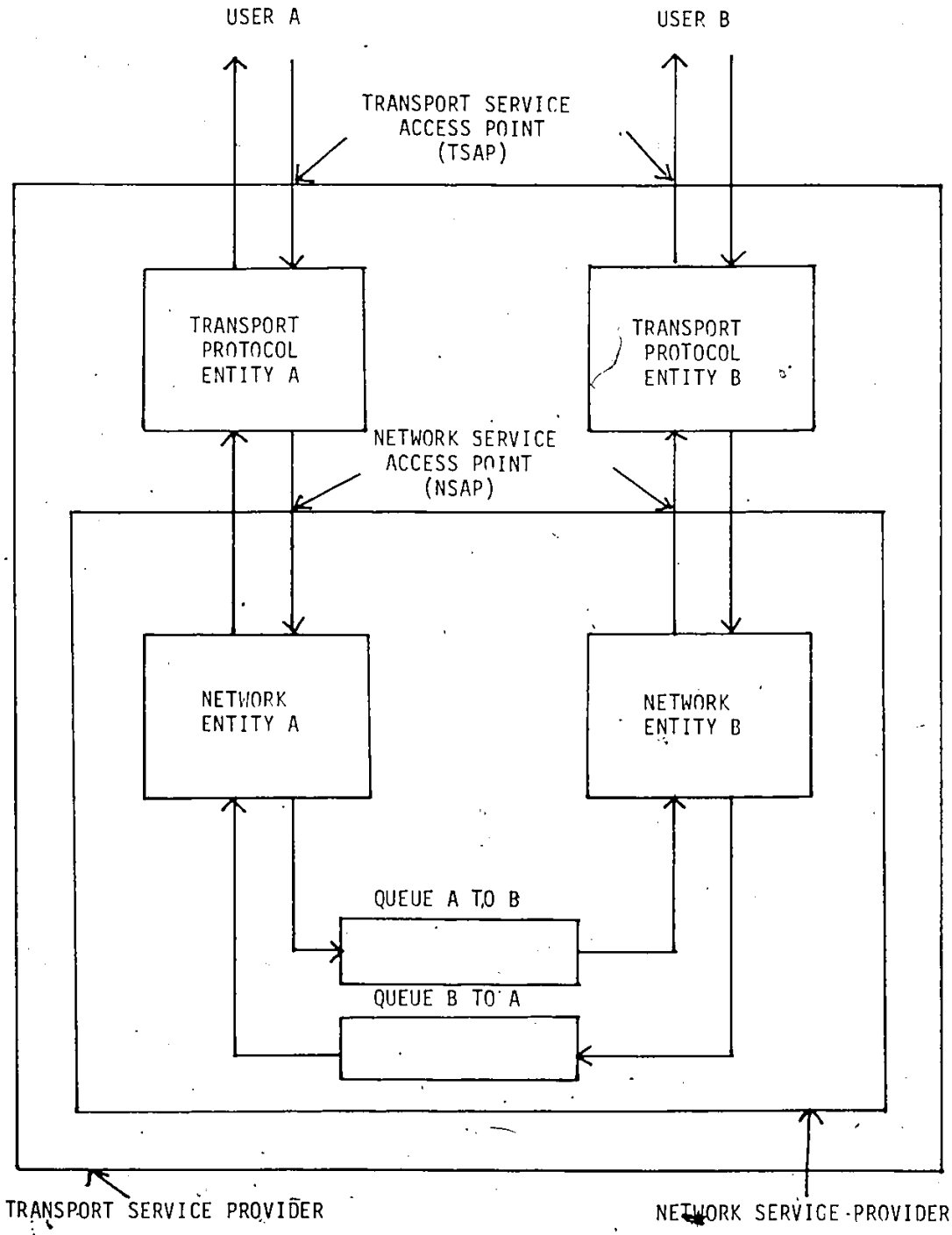


FIG. 3.7

AN IN-DEPTH VIEW OF A TRANSPORT CONNECTION
(WITH PROTOCOL ENTITIES)

3.4 A PASCAL PROTOTYPE FOR CLASS-0 TP SPECIFICATIONS

This section presents the design assumptions made for this implementation, the implementation restrictions, the operating system directives used, structure of various tasks and their communication, window mechanism which the tasks use for sending and receiving data, error handling and a high level view of various modules. It is to be noted here that the operating system and computer used for this implementation is the same as that for the implementation of the prototype of TS specifications, namely RSX-11M on a PDP 11/34.

The impact of layered network architecture of the ISO Reference Model for Open System Interconnection is obvious in this implementation. The implementaion scheme of Fig.3.8 shows a separate Transport Layer and a separate Network Layer. User A and User B can be attributed to Session Layer Entities.

A summary of the Transport Service Primitives used in this implementation alongwith their parameters is given in Table 3.3. A summary of the TPDU's is given in Table 3.4 and a summary of the Network Service Primitives is given in Table 3.5. It is to be noted that the Transport Service primitives and the TPDU's for Transport Expedited Data are not given in the Tables 3.3 and 3.4. because the prototype is for Class-0 TP specifications only which do not support Expedited Data option.

The Pascal routines for User A's side are identical to those for User B's side. Therefore, we present a description of these routines and their Pascal code for User A's side only. This would help in keeping the volume of the code small. Description of the routines for the prototype of TP specifications is given in Appendix C. Pascal code for these routines is given in Appendix D.

PRIMITIVE	DEFINITION	PARAMETERS
TCREQ/TCIND	TRANSPORT CONNECT REQUEST/INDICATION	CALLED ADDRESS, CALLING ADDRESS, EXPEDITED DATA OPTION, QUALITY OF SERVICE.
TCRES/TCCON	TRANSPORT CONNECT RESPONSE/CONFIRM	RESPONDING ADDRESS, QUALITY OF SERVICE, EXPEDITED DATA OPTION.
TDREQ	TRANSPORT DISCONNECT REQUEST	TS USER DATA
TDIND	TRANSPORT DISCONNECT INDICATION	DISCONNECT REASON, TS USER DATA.
TNODR	TRANSPORT NORMAL DATA REQUEST	TS USER DATA.

TABLE 3.3

TRANSPORT SERVICE PRIMITIVES

TRANSPORT PROTOCOL DATA UNIT(TPDU)	DEFINITION
CR	CONNECT REQUEST
DR	DISCONNECT REQUEST
CC	CONNECT CONFIRM
DT	NORMAL DATA
ER	ERROR

TABLE 3.4

TRANSPORT PROTOCOL DATA UNITS (TPDU's) USED

PRIMITIVE	DEFINITION	PARAMETERS
NCREQ/NCIND	NETWORK CONNECT REQUEST/INDICATION	———
NCRES/NCCON	NETWORK CONNECT RESPONSE/CONFIRM	———
NDREQ/NDIND	NETWORK DISCONNECT REQUEST/INDICATION	———
NNODR/NNODI	NETWORK NORMAL DATA REQUEST/INDICATION	NS USER DATA

TABLE 3.5

NETWORK SERVICE PRIMITIVES

3.4.1 Design Assumptions

The following assumptions were made for the design of the prototype for Class-0 TP specifications :

1. The TP specifications are assumed to support only a single transport connection between two transport users. This does not in anyway restrict testing of the connection establishment, Transport data transfer and Transport Disconnection procedures of the Class-0 TP specifications between two Transport users which is precisely the spirit behind developing a prototype for the Class-0 TP specifications. However, in reality for every TCREQ, there is a unique Transport connection between two users.
2. The Network Layer has been modelled by two Network transducers NTRNSDA and NTRNSDB. A simple Network Connection Establishment procedure is assumed. NTRNSDA and NTRNSDB were developed by the experience gained during development of the prototype for TS specifications (see section 3.2) and they behave according to the state diagram of Fig.3.2. This is the reason why the Network Transducers are so similar to the prototype of the TS specifications. They start transfering TPDU's when they are in Data Transfer Ready state. When a Transport Protocol Entity requests a network connection from its respective Network Transducer, the transducer provides that connec-

tion with minimum negotiation with its peer network entity. Therefore, it is assumed that no error situation will occur in the Network Service Provider while establishing a network connection. This is reasonable for the purposes of building a fast prototype for Class-0 TP specifications because the concentration is on the Transport connection rather than detailed negotiation of the Network Connection.

3. It is also assumed that a single Network Connection is possible between User A's side and User B's side. This was done to have a simple Network Service Provider. Emphasis of this exercise has been given to the prototypes of the Class-0 TP specifications i.e Protocol Entities (TPRENTA and TPRENTB). Sophistication of the Network Service Provider is not an objective of this exercise.
4. It is assumed that there is a Network Normal Data queue item 'i' which has a User Data field in which it can carry a single TPDU between the Network Transducers NTRNSDA and NTRNSDB at a time.
5. The action taken by the Transport Protocol Entity in case of receipt of an ER TPDU in OPEN state is not shown in the original state table for Class-0 TP specifications, [24]. The prototype for Class-0 TP specifications handles this by sending a NDREQ to the Network Service Provider and goes to CLOSED state.

6. The receipt of any TPDU other than the ones shown in Table 3.4 is handled by the Transport Protocol Entity by sending a ER TPDU and going to CLOSED state, (see Table 3.2 (b)), [24].

7. The six tasks i.e TERMA, TPRENTA, NTRNSDA, TERMB, TPRENTB and NTRNSDB have not been synchronized because the prototype for TP specifications is meant for laboratory testing in its present form and the testers at both user sites can coordinate the user inputs to avoid deadlock situations such as the one presented in section 3.2. However, these tasks can be synchronized with the help of six global event flags so that simultaneous Transport Connect Requests or Transport Disconnect Requests by User A and User B would not lead to a deadlock or unpredictable system behaviour. It should be noted at this point that even without synchronization of the six tasks in the manner suggested above, two way simultaneous data transfer is possible.

8. A single TPDU is carried by one Network Protocol Data Unit at a time. A single TPDU carrying capability is assumed because it provides the facility to transfer TPDU's between NTRNSDA and NTRNSDB. Sophistication of the Network Layer Data transfer is not an objective here and it does not restrict testing the Class-0 TP prototype's capability to send or receive TPDU's over a simple Network Connection.

9. In case an ER TPDU is to be sent, the Protocol Entity returns the (received) invalid TPDU upto and including the 'first' erroneous parameter to its peer Protocol Entity. This has not been explicitly mentioned in the Class-0 TP specifications, [24]. Therfore, we assume that the error detection of a TPDU by the protocol entity should not proceed after the first error has been discovered.

3.4.2 Implementation Restrictions

Since the executable TP sepecifications is meant to be a prototype, we have not attempted to remove all implementation limitations. These limitations do not invalidate the Class-0 TP specifications in anyway and they do not restrict or limit the testing to be performed on the prototype. These implementation restrictions are :

1. The user primitives such as TCREQ, TDREQ, TCRES are contained in files with file name set to primitive name and file type set to TXT e.g TCREQ.TXT. Each file contains the appropriate parameters for the respective primitive. This has been done so that the user can edit these files and insert the relevant parameter values. Therefore, when the user wishes to send a primitive, he only has to key in the file name and parameter information is read from that file. For primitives such as TNODR, the user is prompted

for user data from the terminal. These user primitive files must be present in the users file directory.

2. The users must always have a video terminal which is compatible with the Digital VT100 type terminal. If this is not so, then the procedures for I/O would have to be modified.
3. Both users must have different User Identification Codes (UIC's) because there are routines which have the same name for both users but have different addresses for the sender and receiver tasks.
4. User data should consist only of printable characters. Therefore, characters such as <ESC> characters are not permissible. Strings upto 118 characters are allowed. Strings longer than this will be truncated to 116 characters. The CRT terminal buffer should also be greater than or equal to 116 characters. Empty strings (only a <CR>) are allowed as user data because the user might not want to send user data even after he/she typed in TNODR.

3.4.3 Additional RSX-11M Directives Used

The executable TP specifications use the SEND DATA and RECEIVE DATA system directives which were used in the executable TS specifications also. These directives carry the control primitives of the Network Service Provider such as NCREQ, NCRES, NCIND etc. which are strings, five characters

long. To exchange data larger than 26-bytes in executable TP specifications, the following system directives are used :

1. SEND BY REFERENCE

This directive instructs the system to queue FIFO a packet containing reference to a memory region (a window mapped on a region) for the destination task. Thus, permitting large blocks of data to be transferred from one task to another. This window also contains a 16 byte block that can be used to send additional information.

Declaration of this directive in the calling Pascal routine is as follows :

```
Procedure sref(var tsk : integer;
               var efn : integer;
               var swdb : wdbb;
               var isrb : ddd;
               var ids : integer);extern(fortran);
```

where :

a) tsk = Destination Task name.

b) efn = Event Flag number.

c) swdb = A 16-byte integer array containing a Window Definition Block.

d) isrb = A 16-byte integer array containing additional information.

e) ids = Directive Status.

2. RECEIVE BY REFERENCE

This directive dequeues the next packet from the Receive by Reference queue of the receiver task. The directive exits if the Receive by Reference queue is empty, [32].

The declaration of the directive in the calling Pascal routine is as follows :

```
Procedure rref(var rwdb : rwdbb;
               var isrb : rddd;
               var ids  : integer);extern(fortran);
```

where :

a) rwdb = A 16-byte integer array containing a Window Definition Block.

b) isrb = A 20-byte integer array to be used as the receive buffer.

c) ids = Directive status.

In addition to the system directives described above, three more RSX-11M directives were used namely, CRAW, CRRG, DTRG. CRAW creates an address window for a task, CRRG creates a dynamic memory region for a task and attaches the region to the task and DTRG simply detaches the attached region from a task, [32].

3.4.4 Window Mechanism

Outlines of the window mechanism and memory region allocation, attachment to a task and detachment by the task is given below. Details can be found in [32] . A window is a part of the memory region whose size and identification is defined by the programmer in the window definition block. This window definition block is sent to the receiver task in the system directive SEND BY REFERENCE. Windows were very useful for our purposes since Network Service primitives and Transport Service primitives were sent and received by reference in these windows.

Whenever a task has to send or receive by reference, it creates a memory region and attaches itself to it. All addressing is done through windows (maximum of seven) in that region. These windows can be mapped (read) or written upon.

Following are the steps a task has to go through in order to send a large amount of data by reference :

1. It creates a memory region for itself and attaches to it.
2. It creates a window in that region.
3. It puts data in that window.
4. It sends the window by reference.

The steps necessary to receive a large amount of data by reference are as follows :

1. The receiver task creates a region and attaches to it.
2. It then receives window by reference (the task is now attached to the sender's region).
3. It accesses data from the window.
4. It detaches from the region.

3.4.4.1 Circular Window Queue

The executable TP specifications employ a circular window queue which can have up to a maximum of seven windows for successive messages to be queued in the region to which the task is attached. This has been done also to prevent loss of data in the case when the sender and the receiver task change their window numbers at the same time. Therefore, when a window is sent by reference, the next window (e.g window 2 after window 1 or window 1 after window 7) is filled with the next set of information. This circular window mechanism ensures no overlap of windows, consequently, no distortion of data.

3.4.4.2 Region Protection

Presently, there is no protection on the memory regions attached to the tasks. As an extension to this implementation, read/ write priveleges can be given to certain tasks to read/write on some other task's region. Details on Region Protection can be found in Section 3 of [32].

3.4.5 Task Structure and Communication

A structural diagram showing the interaction of various tasks is presented in Fig.3.8. There are six tasks which run independently. A brief description of these tasks is given below :

1. TERMA : This task continuously reads the requests from the User A and sends them to the TP Entity A (abbrev. TPRENTA).
2. TPRENTA : This is the Transport Protocol Entity A of Fig.3.7. It receives User A's requests and processes them according to the Class-0 TP specifications given in Tables 3.2 (a), 3.2 (b), 3.2 (c) and 3.2 (d). It also receives TPDU's from its peer entity on User B's side and processes them also according to the above mentioned Tables. This task can send and receive small amounts of data (upto 26-bytes) using the SEND DATA and RECEIVE DATA system directives and large amounts of data using SEND BY REFERENCE and RECEIVE BY REFERENCE system directives (see section 3.4.3).
3. NTRNSDA : This is the Network Entity A of Fig.3.7. It communicates with its peer Network Entity on User B's side and establishes a Network connection. NTRNSDA is then able to carry TPDU's in the User Data field of the Network Normal Data queue items to User B's side. These queue items when received by Network Entity B are disintegrated and the TPDU received is sent to

Protocol Entity B (abbrev. TPRENTB) as a NNODI Primitive. NTRNSDA has been developed based on the experience gained from building the prototype for TS specifications in section 3.2. It behaves according to the state diagram of Fig.3.2. This task is capable of sending and receiving small amounts of data using the SEND DATA and RECEIVE DATA system directives and sending and receiving large amounts of data using the SEND BY REFERENCE and RECEIVE BY REFERENCE system directives (see sections of this chapter on System Directives used). It is to be noted here that NTRNSDA negotiates a Network connection with NTRNSDB first. Only then it starts carrying TPDU's to and from TPRENTA and TPRENTB.

4. TERMB : This is the peer task of TERMA and continuously reads requests from User B.
5. TPRENTB : This is the peer task of TPRENTA and performs identical functions but for User B.
6. NTRNSDB : This is the peer task of NTRNSDA and performs identical functions but for User B.

An example of communication of the six tasks is given below in point form. This example describes the Transport Connection Establishment procedure in the perspective of task communication. It is to be noted that this example does not involve the synchronization issues among various tasks as pointed out in the design assumptions. However,

these tasks can be synchronized by the use of global event flags and by giving priorities to the various tasks over the others to receive as was done in the prototype for TS specifications.

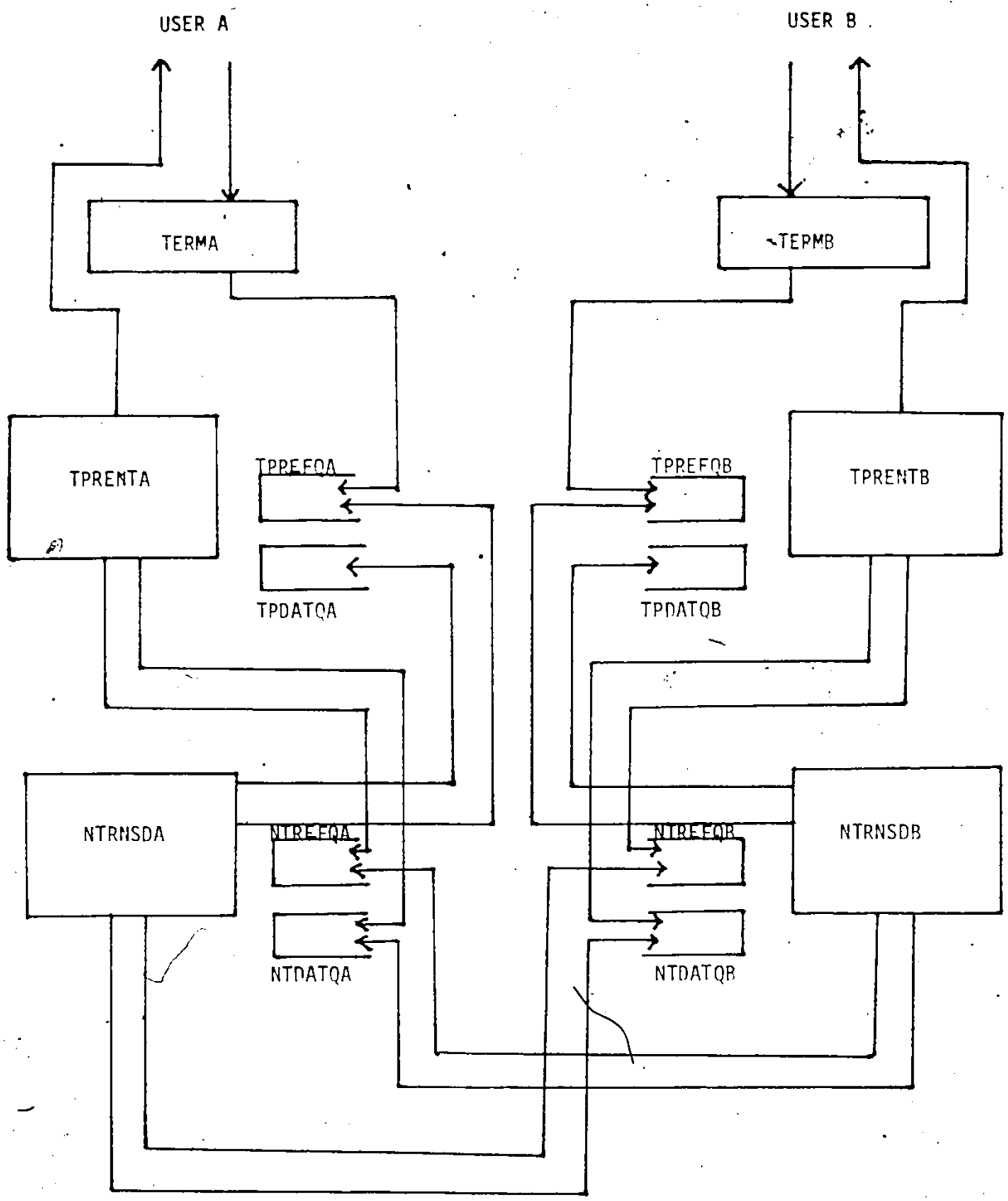


FIG. 3.8

INTERACTION OF TASKS
TRANSPORT PROTOCOL SPECIFICATIONS
IMPLEMENTATION SCHEME

Assume that tasks TPRENTA and TPRENTB are in CLOSED state (see Tables 3.2 (a) and 3.2 (b)) Assume that there is no Network Connection available i.e NTRNSDA and NTRNSDB are in IDLE state. Suppose that User A wishes to initiate a Transport Connection. The following sequence of events takes place, (see Fig.3.8) :

1. User A punches in TCREQ. The task TERMA reads the parameters of the TCREQ primitive from the file TCREQ.TXT and uses the SEND BY REFERENCE directive to send this information to task TPRENTA which is written on the task queue TPREFQA.
2. TPRENTA is in CLOSED state. It receives the TCREQ primitive using the RECEIVE BY REFERENCE directive. It now checks if there is a Network Connection available. There is none. Therefore, it sends a NCREQ to NTRNSDA using the SEND DATA primitive which is written on the task queue NTDATQA. It then prepares a CR TPDU and changes its state to WFNC.
3. NTRNSDA receives the NCREQ, sends a queue item 'q' to NTRNSDB which is written on the task queue NTDATQB using the SEND DATA directive. It then changes its state to OUT-CONNECT PENDING.
4. NTRNSDB receives the queue item 'q' using RECEIVE DATA directive, sends a NCIND to task TPRENTB using the SEND DATA directive which is written on the task queue TPDATQB and changes its state to IN-CONNECT PENDING.

5. Task TPRENTB is in CLOSED state. It receives the NCIND from NTRNSDB using the RECEIVE DATA directive, sends a NCRES to NTRNSDB which is written on the task queue NTDATQB using the SEND DATA directive and stays in CLOSED state.

6. Task NTRNSDB receives the NCRES from TPRENTB using the RECEIVE DATA directive and sends a queue item 's' to NTRNSDA using the SEND DATA directive which is written on the task queue NTDATQA and changes its state to DATA TRANSFER READY state.

7. Task NTRNSDA receives the queue item 's' using the RECEIVE DATA directive and sends a NCCON to TPRENTA which is written on the task queue TPDATQA indicating that a Network connection has been established. NTRNSDA then changes its state to DATA TRANSFER READY state.

8. Task TPRENTA now sends the CR TPDU to task NTRNSDA using the SEND BY REFERENCE directive. This TPDU is sent in the User Data field of the NNODR Network Service Primitive which is written on the task queue NTREFQA. Task TPRENTA then changes its state to WFCC state.

9. Task NTRNSDA receives the CR TPDU using the RECEIVE BY REFERENCE directive and concatenates the CR TPDU into the User Data field of a Network Normal Data queue item and sends the queue item to NTRNSDB using

SEND BY REFERENCE directive which is written on the task queue NTREFQB.

10. NTRNSDB receives the Network Normal Data queue item from NTRNSDA using the RECEIVE BY REFERENCE directive and separates the CR TPDU from it's User Data field. It then sends the CR TPDU to TPRENTB as a parameter of the NNODI Network Service Primitive using the SEND BY REFERENCE directive. This NNODI is written on the task queue TPREFQB.
11. TPRENTB receives the NNODI using the RECEIVE BY REFERENCE directive. It then separates the CR TPDU from the NNODI Network Service primitive. Then, TPRENTB maps the information in the CR TPDU on a TCIND primitive and writes all of this information on User B's terminal. It then changes its state to WFTRESP.
12. User B now types in TCRES. The task TERMB reads the information about all parameters of TCRES primitive from the file TCRES.TXT and sends it to task TPRENTB using SEND BY REFERENCE directive which is written on the task queue TPREFQB.
13. TPRENTB receives TCRES with parameters using RECEIVE BY REFERENCE directive and, maps the information on a CC TPDU, sends it to NTRNSDB using the SEND BY REFERENCE directive in the User Data field of the Network Service Primitive NNODR. This is written on

the task queue NTREFQB. It then changes its state to OPEN state.

14. NTRNSDB receives the CC TPDU, concatenates it into the User Data field of the Network Normal Data queue item and sends the queue item to NTRNSDA using the SEND BY REFERENCE directive. This queue item is written on the task queue NTREFQA.
15. NTRNSDA receives this queue item, separates the CC TPDU from its User Data field and sends the CC TPDU to TPRENTA using SEND BY REFERENCE directive as a parameter of the NNODI Network Service Primitive. This primitive is written on the task queue TPREFQA.
16. TPRENTA separates the CC TPDU from the NNODI primitive and maps this information on the TCCON primitive. It then writes all of this information on User A's terminal and changes its state to OPEN state. The Transport Connection is now OPEN.

This was a simple example of Transport Connection establishment. The executable specifications actually allow any of the two users to initiate a Transport Disconnection procedure by typing in TDREQ and providing additional information for disconnection. This is handled exactly in accordance with the TP specifications given in Tables 3.2 (a), 3.2 (b), 3.2 (c) and 3.2 (d).

Once Transport Connection has been established, Transport data can be transferred on both sides using the TNODR primitive. This transfer of data is sent and received using the SEND and RECEIVE BY REFERENCE directives. The Network Normal Data primitives carry the Transport Data TPDU's (DT TPDU's) between TPRENTA and TPRENTB.

3.4.6 Error Handling

There are two types of errors that the Prototype handles :

1. Erroneous user primitive name. This error is handled by giving an error message to the user and prompting him for a legal primitive name.
2. Erroneous TPDU received from peer protocol entity. This error is handled by transmitting an ER TPDU to the peer TP Entity. This ER TPDU contains the erroneous TPDU upto and including the byte in which the first error occurred and the cause for rejection of the TPDU as per [24]. The ER TPDU is displayed on the terminal of the user who receives it with all its parameters in a user friendly manner. Errors in the received TPDU's are detected by the Transport Protocol Entities using the Procedure INSPTPDU.

System errors arise occasionally. The prototype has no real facilities to deal with these problems. Two types of such errors are described below :

1. Errors arising from system directives :

In case of receipt of error codes from the system directives, the task halts using the HALT primitive of Swedish Pascal and has to be started again if further communication is desired with it. These errors are detectable in that the behaviour of the task is predictable. Occurence of such an error in a prototype is tolerable because it can only disrupt the testing process and can alert the designers if they are to use the same system directive in the design and implementation of the real protocol.

2. Implementation errors :

Some non-detectable errors were also discovered during the implementation process. These errors included :

a) Run-time error if the keyword EXTERN was missing from the external procedure declaration but the error message would not report this error.

b) Run-time error if the declared Stack or Heap size was too small. It was observed that the run-time error message did not report the real cause. The Stack or Heap size can be increased using the (/INC=nnnn) run-time option.

3.4.7 High Level Structure of Modules

Fig.3.9, Fig.3.10, Fig.3.11 show the high level flow diagrams of the tasks TERMA, TPRENTA and NTRNSDA respectively. The three tasks for Side B i.e TERMB, TPRENTB and NTRNSDB are identical to those of Side A. Their flow diagrams are therefore ommitted.

These diagrams give a high level view of the various procedures, functions and system directives used in each task and their relation to eachother. It is to be noted here that all procedures and functions are external to their calling modules and have been separately compiled in different files.

The name, function, usage, input parameters, output parameters, exit conditions etc. of each procedure can be found in Appendix C.

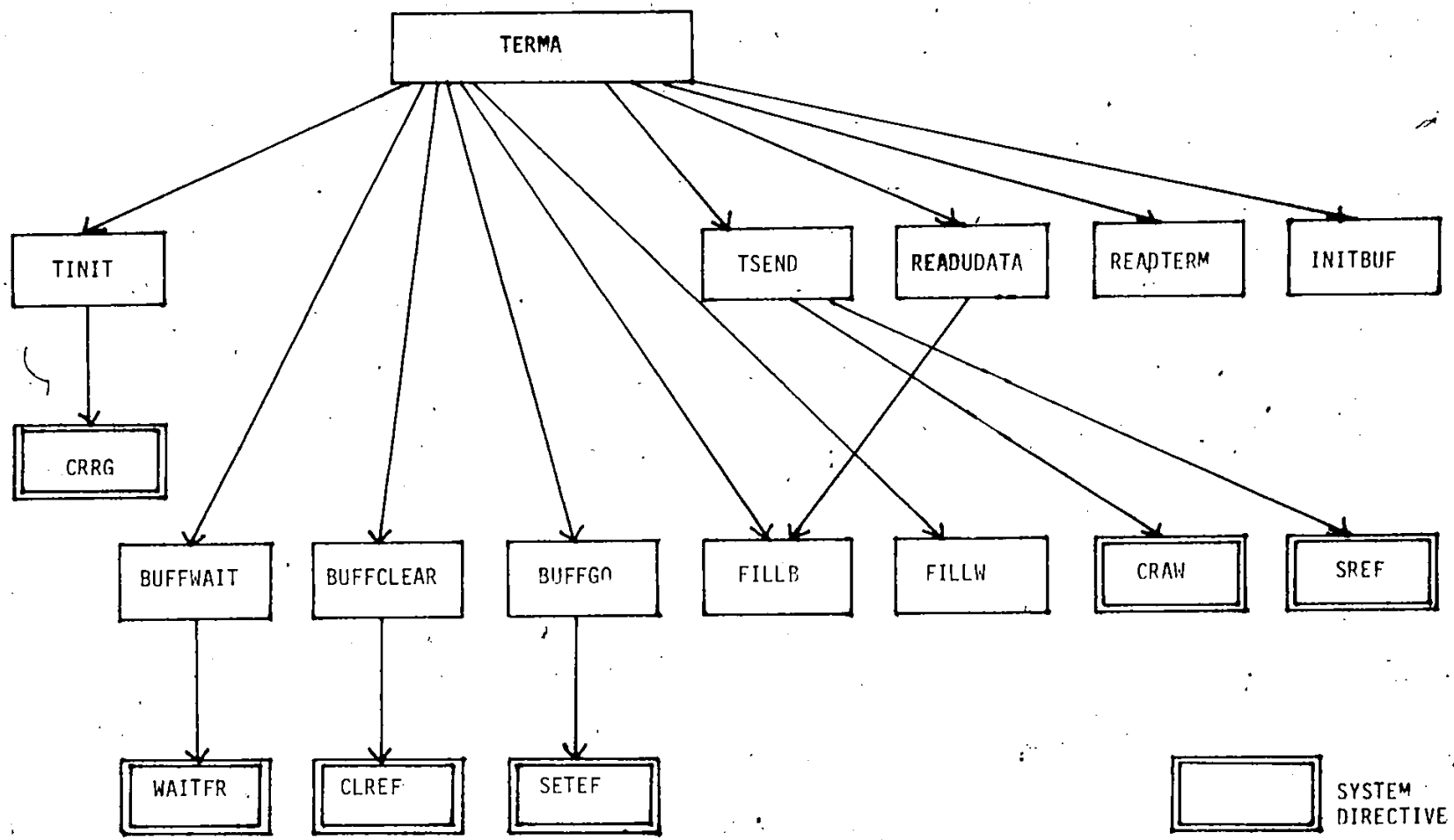


FIG. 3.9

HIGH LEVEL STRUCTURE OF TASK TERMA
(TP SPECIFICATIONS)

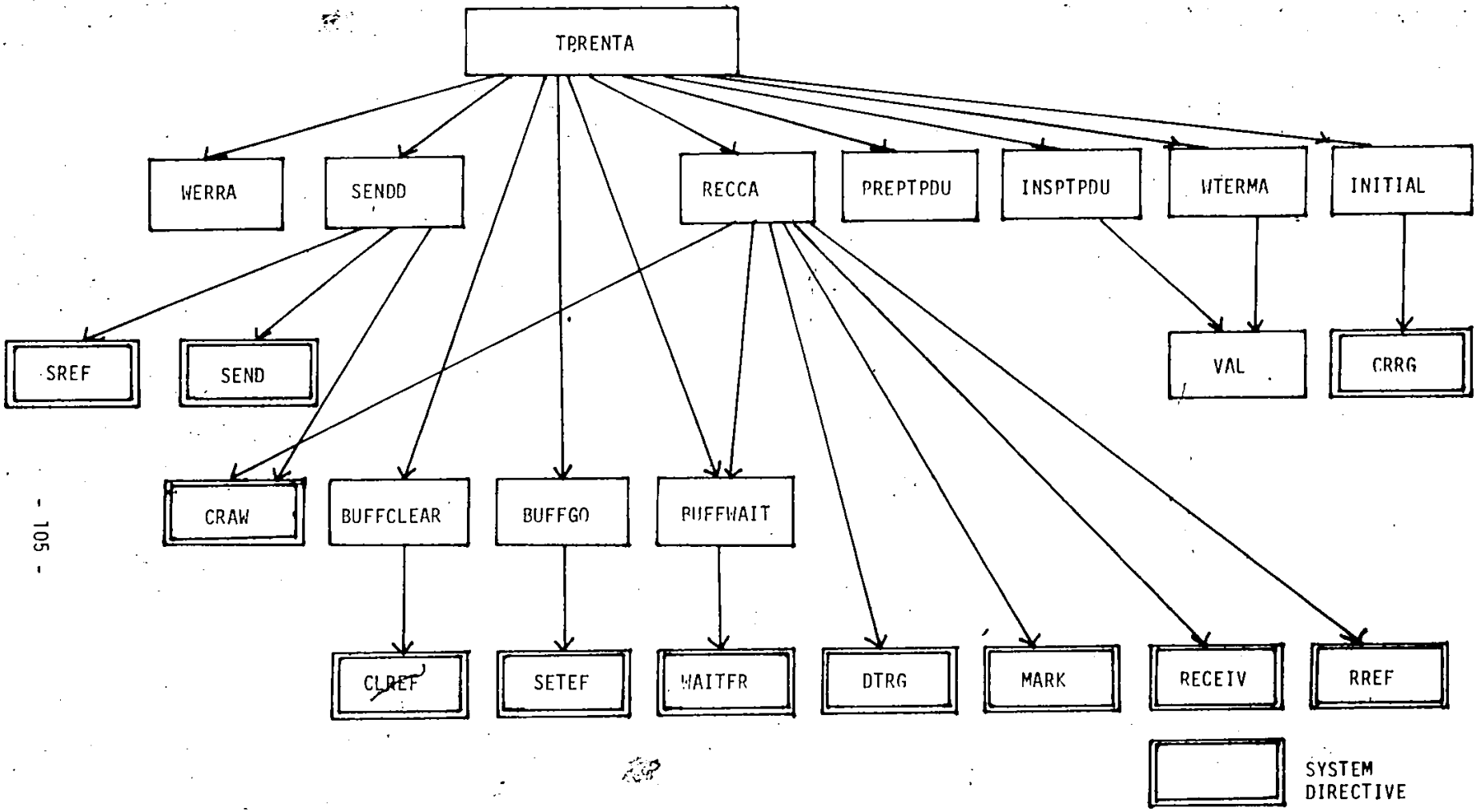


FIG. 3.10

HIGH LEVEL STRUCTURE OF TASK TPRENTA.
(TP SPECIFICATIONS)

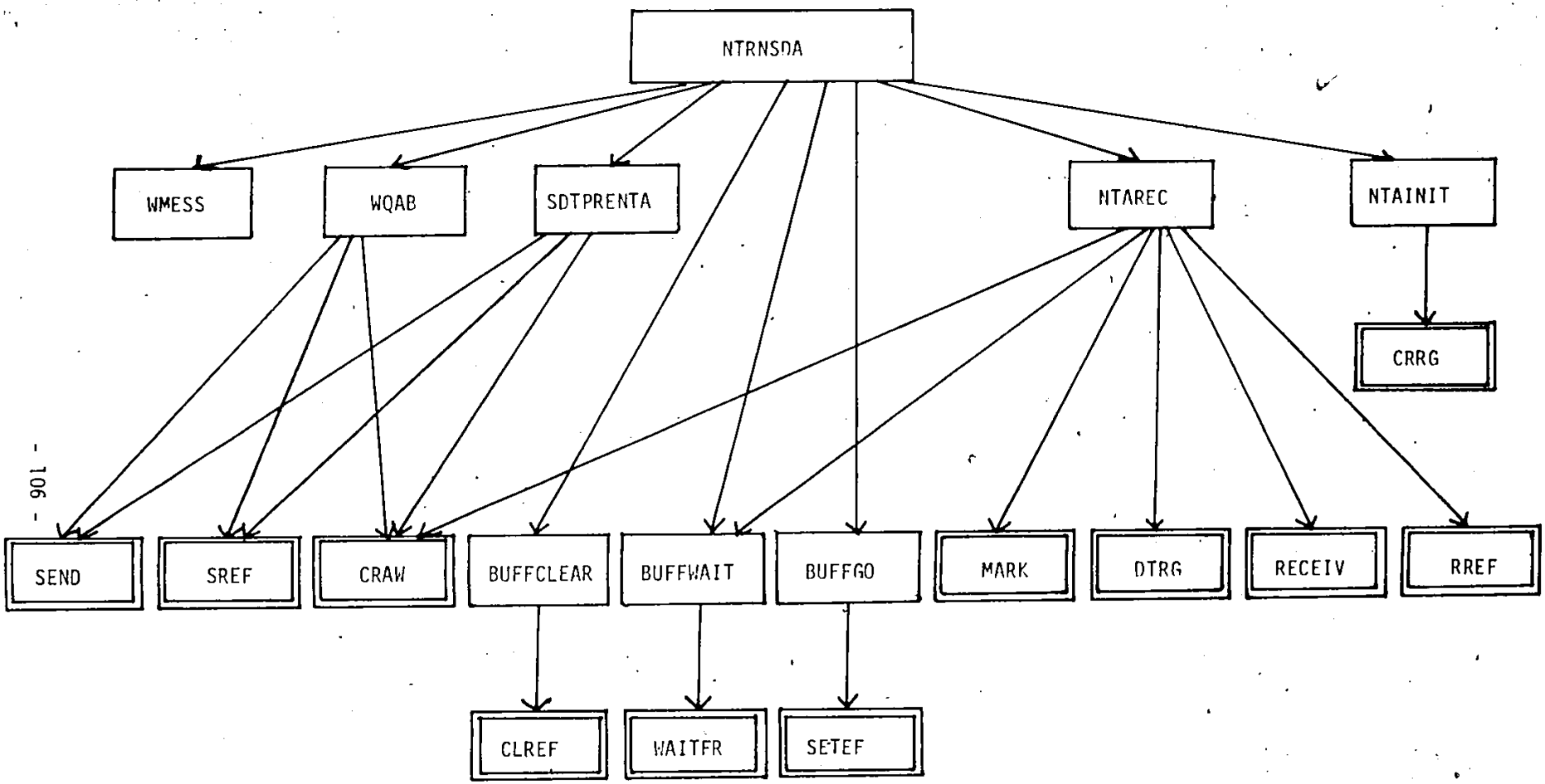


FIG. 3.11

HIGH LEVEL STRUCTURE OF TASK NTRNSDA
(TP SPECIFICATIONS)

3.5 SOFTWARE DESIGN CONSIDERATIONS AND QUALITY ASSURANCE

This section deals with the software design considerations and software testing strategy adopted during the development of the prototypes of TS specifications and TP specifications.

3.5.1 Software Design Considerations

The main considerations taken into account while designing the prototypes for TS and TP specifications are given below. These considerations are the same for both prototypes in principle. Therefore, they have been illustrated with reference to the prototype for TP specifications only.

Modularity :

Design of both prototypes is modular. For example, the prototype for TP specifications has three tasks on each user's side i.e for user A, there is TERMA, TPRENTA and NTRNSDA. The design is modular in that :

1. Each of these tasks have external procedures and functions (see Fig.3.9, Fig.3.10, Fig.3.11) for high level view of these tasks and their external routines.
2. No global variables are used for any task.

Portability :

The prototypes have been designed to promote portability as follows :

1. All procedures and functions of every task are external to the calling routines. It is assumed that the operating system on which the prototype will be used in future supports the use of external routines. A view of the procedures and functions is given in the high level flow diagrams of Fig.3.5 and Fig.3.6 for the prototype of TS specifications. Similar diagrams for the tasks of the prototype of TP specifications are given in Fig.3.9, Fig.3.10 and Fig.3.11.
2. Whenever a task has to send or receive data using the SEND DATA, SEND BY REFERENCE, RECEIVE DATA or RECEIVE BY REFERENCE directives, it does not call any of these directives directly. Instead, it calls an external sender routine to send data using SEND DATA or SEND BY REFERENCE directives and calls an external receiver routine to receive data using RECEIVE DATA or RECEIVE BY REFERENCE directives. This makes the task portable to other operating systems since only the sender and the receiver routines would have to be modified.

The following points describe the degree of system dependence of the prototypes :

1. The display of information to the users is Digital terminal VT100 dependent.
2. The prototypes require an operating system which allows Fortran I/O from Pascal routines.
3. System directives or corresponding routines should be available for intertask communication.

The author feels that this degree of system dependence is reasonable for prototypes.

Modifiability

The most important point to note is that the prototype for TP specifications is readily modifiable. More routines can be added on to it incrementally if :

1. Multiple Transport connections are to be supported.
2. All errors in an erroneous TPDU received are to be detected.
3. Changes have to be made in the prototype for using it as the error generator for a peer entity under test (EUT) in the testing architectures (see Chapter 4).

3.5.2 Software Testing

This subsection deals with the software scheme adopted in testing prototypes of TS specifications and TP specifications. But, for the sake of brevity, the testing scheme will be presented in the perspective of the prototype of TP specifications. The software testing scheme consisted of the following :

Module testing

User A's side consists of the following tasks (see Fig.3.8) :

1. TERMA.
2. TPRENTA
3. NTRNSDA

User B's side consists of the following tasks :

1. TERMB
2. TPRENTB
3. NTRNSDB

Each procedure of the tasks on User A and User B's side was compiled in separate files. The tasks on User A's side were tested individually. Then, the tasks on User B's side were tested individually according to the scheme given below. Dummy tasks are required in this scheme because the sender task requires the receiver task's name and the system directive halts the sender task if the receiver task is not found and no further testing can be performed before the sender task is started up again.

Task TERMA was tested individually with a dummy task replacing task TPRENTA. It was tested for all kinds of user requests such as TCREQ, TCRES, TNODR and TDREQ with their appropriate parameters. Then task, TPRENTA was tested and tasks NTRNSDA and TERMA were replaced by dummy tasks. TPRENTA was given input from User A and responses from the

dummy task in place of NTRNSDA. The changes in its states were carefully noted by the help of debug statements and matched against the state table given in Tables 3.2 (a), 3.2 (b), 3.2 (c) and 3.2 (d). Then task NTRNSDA was tested and dummy tasks were run in place of NTRNSDB on User B's side and TPRENTA above it. It was tested for successful sending and receiving of Network primitives for establishment of a Network connection.

Tasks on User B's side were also tested individually in a similar way.

Integration testing :

Integration testing was performed in two steps :

1. Integration of modules on each user's side.
2. Integration of all modules.

Integration of modules on User A's side was done by running TERMA, TPRENTA and NTRNSDA together with a dummy task for NTRNSDB. Similarly, User B's side was tested by running TERMB, TPRENTB and NTRNSDB with a dummy task for NTRNSDA.

Then all six tasks were run together with actual user requests. At this point, a serious problem was encountered :

After having used the system directive RECEIVE BY REFERENCE, a task would not receive any data using the system directive RECEIVE DATA for an arbitrary number of times

and then suddenly receive all data using the RECEIVE DATA directive. The diagnosis of this problem was found to be a compiler error due to which system directives behave in an unpredictable manner in case RECORD type of data structures are passed as variable parameters between procedures. This problem was overcome by passing POINTERS to the RECORD's instead of the RECORD's themselves.

Regression testing

Regression testing was performed using the same sequence of test inputs after the problem with passing the RECORD type was overcome. This was done to determine that the effect of modifications in the tasks was local to the problem encountered and did not affect other functions of the system. This was indeed the case.

The author feels that the prototype for TP specifications developed in section 3.4 can be safely used to check the functional behaviour of the Class-0 TP specifications.

Chapter 4

APPLICABILTY OF PROTOTYPES IN PROTOCOL TESTING ARCHITECTURES

This chapter presents an introduction to some related research in protocol testing architectures, the potential role of prototypes in these testing architectures and a plan to apply the prototypes of Class-0 TP specifications (TPRENTA and TPRENTB) in an elementary configuration for laboratory testing of protocols which is also proposed in this chapter The results of the application of selected test sequences to the prototype TPRENTB are also presented. Some of the protocol testing concepts in this chapter have been proposed by H. Ural and R. L Probert [20].

4.1 INTRODUCTION TO RELATED RESEARCH IN PROTOCOL TESTING ARCHITECTURES

This section presents a brief survey of the models for protocol testing architectures proposed in the literature and the effective use of the prototype for Transport Protocol in these architectures.

4.1.1 Protocol Testing Architectures....A Survey

Several models for protocol testing architectures have been proposed in the literature. Two such models are presented below :

4.1.1.1 NPL Model for Protocol Testing Architectures

The National Physical Laboratory's Model for Protocol Testing Architectures provides the facility to test a Layer N protocol implementation between two open systems, one located at the Client's Site and the other at the Assessment or Laboratory Site, [21], (see Fig.4.1). The Assessment Site consists of the following components :

1. A Test Driver.
2. Layer N protocol implementation + error Encoder/Decoder.
3. Layer N-1 protocol implementation + Exception generator

The Client's Site system consists of a set of similar components as follows :

1. A Test Responder.
2. Layer N Protocol Implementation Under Test (IUT).
3. Layer N-1 Protocol implementation.

There is an underlying Layer N-2 communication channel common to the Client's Site and the Assessment Site. Details of the above components can be found in [21].

The NPL Model assumes that to test a Layer N protocol implementation, only the protocols and services below Layer N have been implemented and not those above it [21]. This architecture also considers the Implementation Under Test (IUT) to be a black box i.e the details of the internal structure of the IUT are not considered when test sequences are selected.

The NPL model has been adopted by several research groups in different countries to develop test architectures suited to their particular needs for certification purposes, [21].

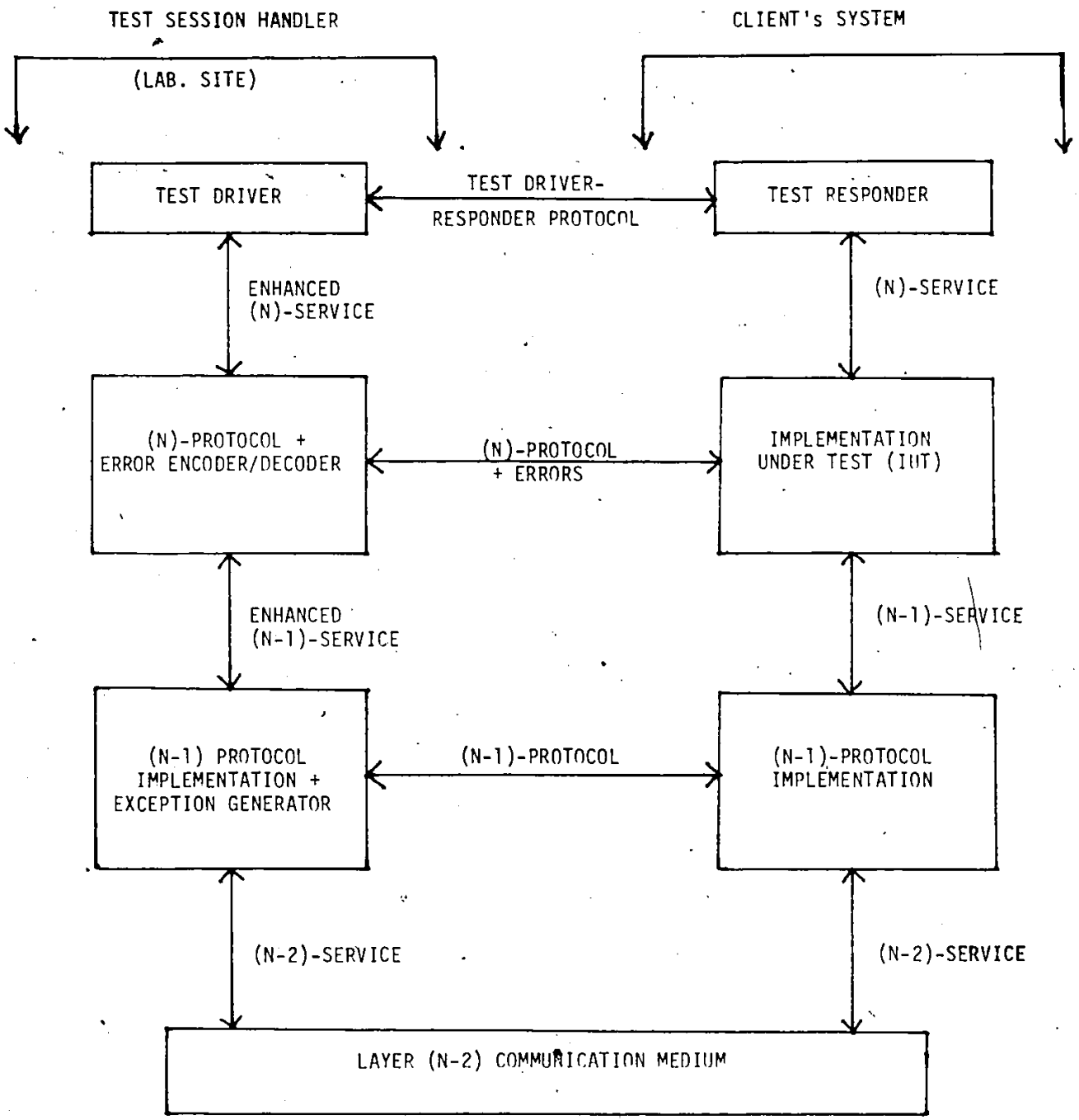


FIG. 4.1

NPL MODEL FOR PROTOCOL TESTING ARCHITECTURE

4.1.1.2 Other Testing Architectures

A model based on two open systems interacting over a network connection has been proposed in [20]. This model is shown in Fig.4.2. The basic components of this model are :

1. A Laboratory Site Test Manager.
2. A Laboratory Site Test Supervisor.
3. An Emulator for peer of entity under test.
4. A communication medium.
5. The Entity Under Test (EUT) on the Client Site.
6. The Client Site Test Supervisor.

Details of these components and their configuration for various test architectures can be found in [20].

Logical architectures for protocol testing have been developed for Laboratory (Assessment site) testing and for Remote (Client's Site) testing by expanding the model incrementally. The Laboratory Testing Architecture needs an emulator for the communication medium and the Entity Under Test is in the laboratory, whereas in remote testing the Entity Under Test is at the Client's Site and is tested over a real communication medium, [20].

The effectiveness of the black-box testing approach has been enhanced in this model by incorporating adaptations of some testing strategies such as Functional testing, Boundary-Value Analysis and Error-Based Testing, [20].

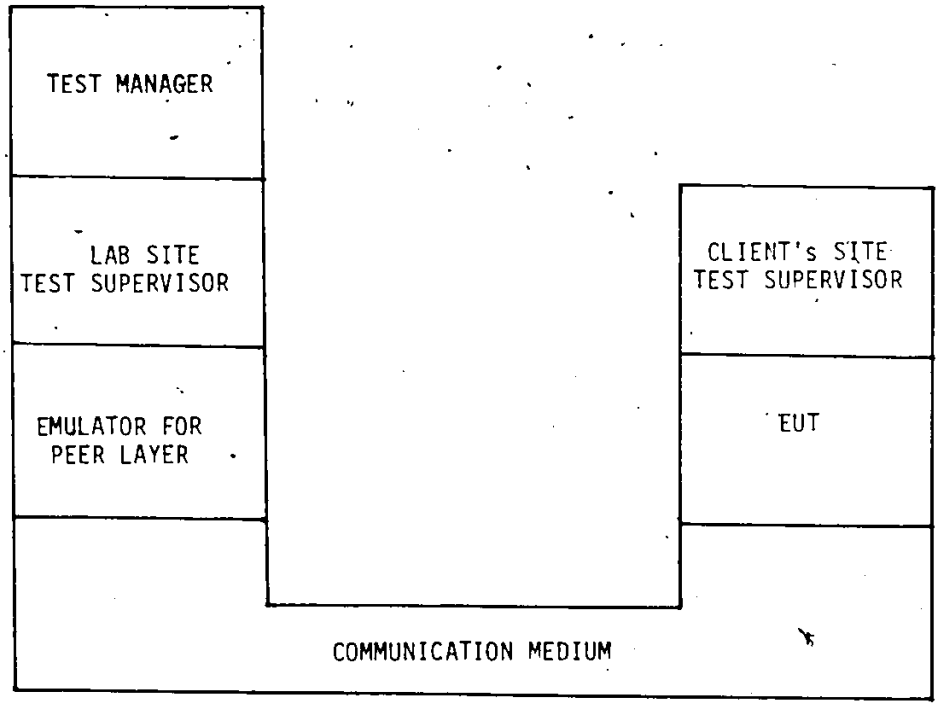


FIG. 4.2

MODEL FOR PROTOCOL TESTING ARCHITECTURES

4.1.2 Effective Role of Prototypes in Testing Architectures

All of the testing architectures described above have a component called an Entity Under Test (EUT) or an Implementation Under Test (IUT). This component can be :

1. An actual protocol implementation or
2. Executable Specifications of Layer N protocol in the form of a prototype.

The Entity Under Test can be either an actual implementation of the protocol when an existing protocol implementation is to be tested or it can be a new implementation to be tested. In both cases, the protocol has been implemented. If basic design errors are detected, a lot of effort and money could be expended in redesigning and recoding the software.

This leads us to the concept of high-level software testing, which calls for detecting errors at an earlier stage of the development process i.e before the actual implementation. This concept involves the migration of software testing methodologies from the implementation phase upward into the design and specification phases, [33].

An inexpensive way of testing the desirability of the ultimate product (e.g a protocol implementation) is to develop a fast prototype which has all of the functional features of the specifications. As suggested by H. Ural and R. L Pro-

bert in discussions, a prototype can be the laboratory site emulator for peer layer under test. It has also been suggested during these discussions that use of prototypes at laboratory site and at the client's site allows "live" testing of specifications.

The fast prototype developed for Class-0 TP specifications (on User B's side) is used as the Entity Under Test (EUT) for running selected test sequences through it and its behaviour is observed. Changes are made to the prototype for User A so that it is possible to generate erroneous TPDU's. It should be noted that the selection of test sequences is specification-based. By applying these selected test sequences to the fast prototype, suitability of the TP Specifications can be evaluated. The protocol can be actually implemented once confidence has been developed in the prototype.

It was suggested by H. Ural during discussions that fast prototyping approach can be useful in testing the design of other components of the testing architecture such as a Test Supervisor or a Test Manager prior to their actual implementation.

4.2 APPLICATION PLAN

This section presents the test plan, an elementary configuration for laboratory testing of protocols and the resulting impact of this configuration on one of the prototypes for TP specifications. This impact involves changing one of the Transport Protocol Entities to generate TPDU's in error.

Protocol Entity A (task TPRENTA) will be changed such that it can generate erroneous TPDU's. Protocol Entity B (task TPRENTB) will be considered the Entity Under Test (EUT). Erroneous Transport Service primitives will be supplied by the tester on User B's side while erroneous TPDU's can be supplied by the modified protocol entity TPRENTA.

4.2.1 Test Plan

The test plan has three objectives as follows :

1. To detect functional errors of the protocol entity.

   The Transport protocol entity performs the three main functions given below :

   a) Transport Connection Establishment with negotiation between the Transport Service users.

   b) Data Transfer between the Transport Service users.

   c) Connection Release initiated by either of the Transport Service users or the Network Service Provider.

Each of the above functions has some sub-functions to perform such as concatenation and separation of TPDU's, mapping various parameters of user primitives such as TCREQ onto the TPDU's. The selected test sequences test these sub-functions as well.

2. To allow for functional dependencies between protocol functions.

   Functional dependencies will be taken into consideration before applying the test sequences, [34]. The functions of Transport Protocol shown above are dependent on each other. For example, for successful Transport Data Transfer, successful Transport Connection Establishment is necessary and so on. Therefore, we will apply those test sequences to the Transport Protocol Entity which help establish correctness of the Transport Connection Establishment function first. Then we can check the correctness of Transport Data Transfer function of the protocol. This is followed by checking the Transport Disconnection Procedure.

3. To test error detection/reporting capabilities of protocol entity.

   Class-0 Transport Protocol specifications do not require the protocol to recover from protocol errors but error detection and reporting is required. Test

sequences were selected to test the above mentioned error handling capabilities of Transport Protocol Entity. The Transport Protocol Entity is subjected to user requests and TPDU's with invalid parameter codes or values according to the testing configuration in the next subsection.

4.2.2 An Elementary Testing Configuration

An elementary testing configuration was used to show the applicability of fast prototypes to Laboratory testing of protocols. This configuration is shown in Fig.4.3. Functionally, this diagram is the same as that of Fig.3.8. The details of task queues and actual flow of data have been ommitted for the sake of clarity and to highlight the features of this diagram as being an elementary testing configuration for laboratory testing of protocols. It is also shown in this section how these prototypes can be improved and incorporate them in a more sophisticated protocol testing architecture. However, this is beyond the scope of this thesis.

The system shown in Fig.3.8 does not allow User A or User B to directly input to the Protocol Entities (TPRENTA and TPRENTB). In order to enable TPRENTA to generate erroneous TPDU's on demand, it has been modified so that it can accept input from User A's side to generate errors in TPDU's (see Fig.4.3) which can then be sent to the Entity Under Test (TPRENTB). Example test sequences applied to TPRENTB are

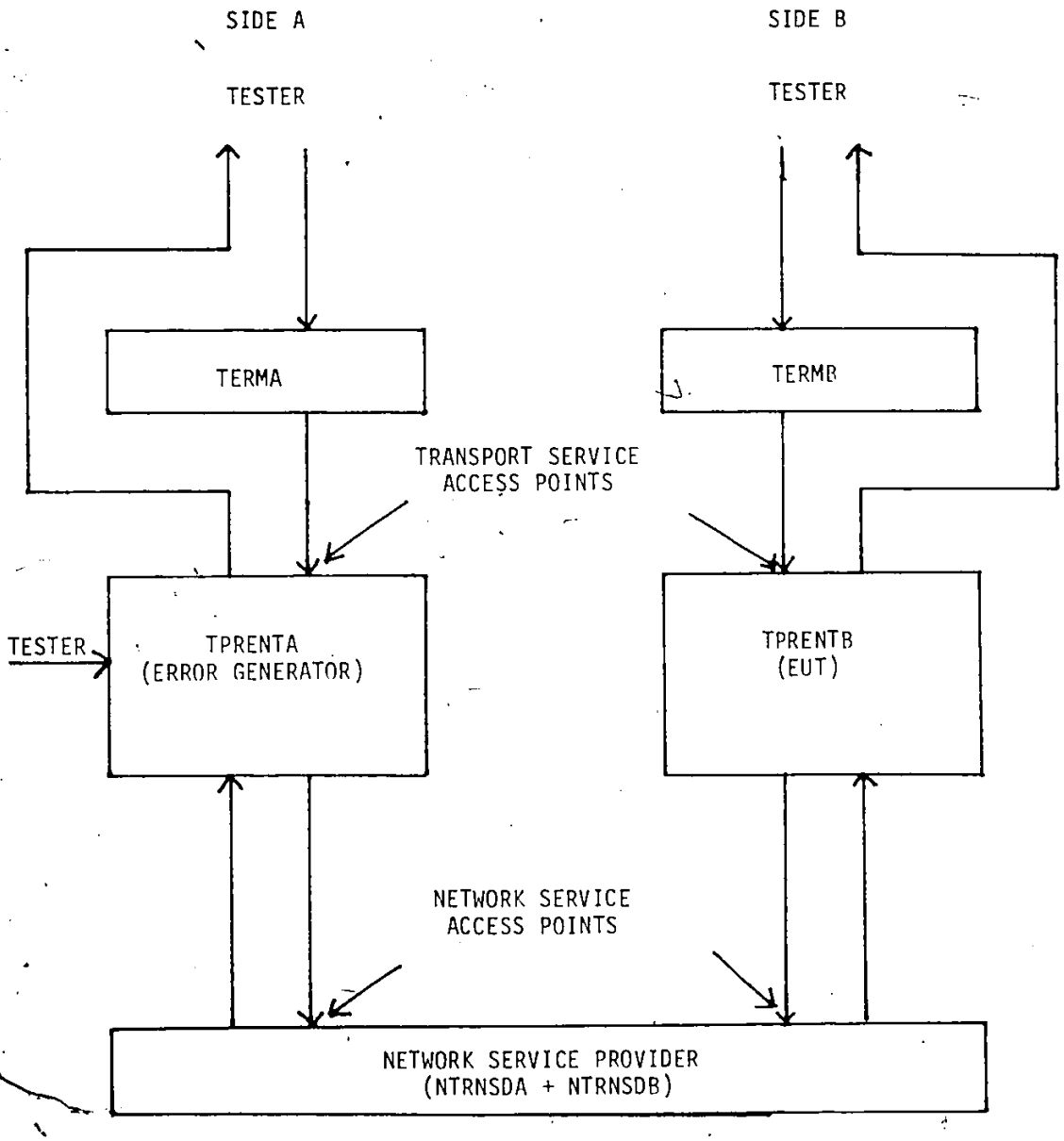
presented. in Figs.4.4 through 4.9. and are discussed in section 4.3.

During testing, the tester has a list of the various fields of different TPDU's. All fields on the list are numbered. The Transport Entity A in the modified form prompts the tester to :

1. Enter the field number of the particular TPDU to be changed.
2. After the tester enters the field number, he/she is prompted for the erroneous value to be entered in this field.

Once the above two steps have been done, the tester can type in 999 to signal TPRENTA to send the erroneous TPDU. The behaviour of TPRENTB in response to this erroneous TPDU can then be observed at the video terminal at which it is running. The tester can also input erroneous user primitives from User B's side and can observe the response of TPRENTB at the terminal on which the task TPRENTB is running. This scheme is simple and it provides the tester with a full capability to input erroneous parameter values and codes in the TPDU's without significantly modifying TPRENTA. The disadvantage is that the testing process is manual. Test sequences have to be input one at a time and the results noted visibly. Also, the tester has to be present at both User A's side as well as on User B's side.

As an alternative testing configuration, this process can be semi-automated by augmenting TPRENTA so that it generates erroneous TPDU's when the tester specifies such an action from User A's side. The erroneous values can be stored in a table. The tester can specify the index to the erroneous parameter value and the type of TPDU e.g CR TPDU and the procedure preparing TPDU's (procedure PREPTPDU of task TPRENTA) can then insert erroneous values into the TPDU. Further automation can be achieved by replacing some of the functions of the human tester by a Test Manager and Test Supervisor programs, [33]. A sophisticated test sequence results reporting scheme can also be implemented. Therefore, the testing configuration of Fig.4.3 can be built upon in an incremental fashion resulting in an implementation of a standard protocol testing architecture for laboratory testing of protocols, [20, 21].



NOTE: THIS DIAGRAM IS FUNCTIONALLY THE SAME AS FIG.3.8 . DETAILS OF TASK QUEUES ARE OMMITTED FOR THE SAKE OF CLARITY.

FIG. 4.3

AN ELEMENTRY TESTING CONFIGURATION
FOR LABORATORY TESTING OF PROTOCOLS

4.3 APPLICATION EXPERIENCE

Test Sequences derived on the basis of the Test Plan of section 4.2 were applied to TPRENTB. The results are summarized in Figs.4.4 through 4.9. The following points should be noted prior to reading these results :

1. Network Transducers NTRNSDA and NTRNSDB of Fig.3.8 have been combined as one Network Service Provider in the testing configuration. It is acceptable to provide only a high-level prototype Network Service Provider since our detailed focus is on the behaviour of the prototypes of Class-0 TP specifications.
2. Test sequences representing only the basic functions of the TP specifications are presented. These results clearly show the applicability of the prototype in an elementary configuration for laboratory testing of protocols. Many more sequences were run but they will not be presented here to avoid repitition.
3. Parameters of the Transport Service primitives, the Network Service primitives and the TPDU's have not been shown in tabulation of the results for the sake of clarity.
4. The results given in Figs.4.4 through 4.9 had to be manually tabulated because the RSX-11M operating system V3.2 on the PDP 11/34 distorts the contents of an output file if two or more tasks write on the same output file simultaneously. Therefore, two tasks

could not be made to write the results of a test sequence on the same output file. This problem can be resolved by writing a sophisticated File Locking Driver program. This program would instruct a task to wait until another task has finished writing on an output file if the same file is to be used as an output file for both tasks. The file locking driver was ommitted because the results could be tabulated legibly after making observations and the implied time to construct the driver could not be cost-justified.

5. The fact that a TPDU is acceptable or not to TPRENTB is indicated by a note saying 'accepts' or 'rejects'.
6. The state that TPRENTB attains after sending / receiving a particular primitive, is written against that primitive e.g '.WFCC.' or '.WFNC.'.
7. The time scale is shown horizontally.

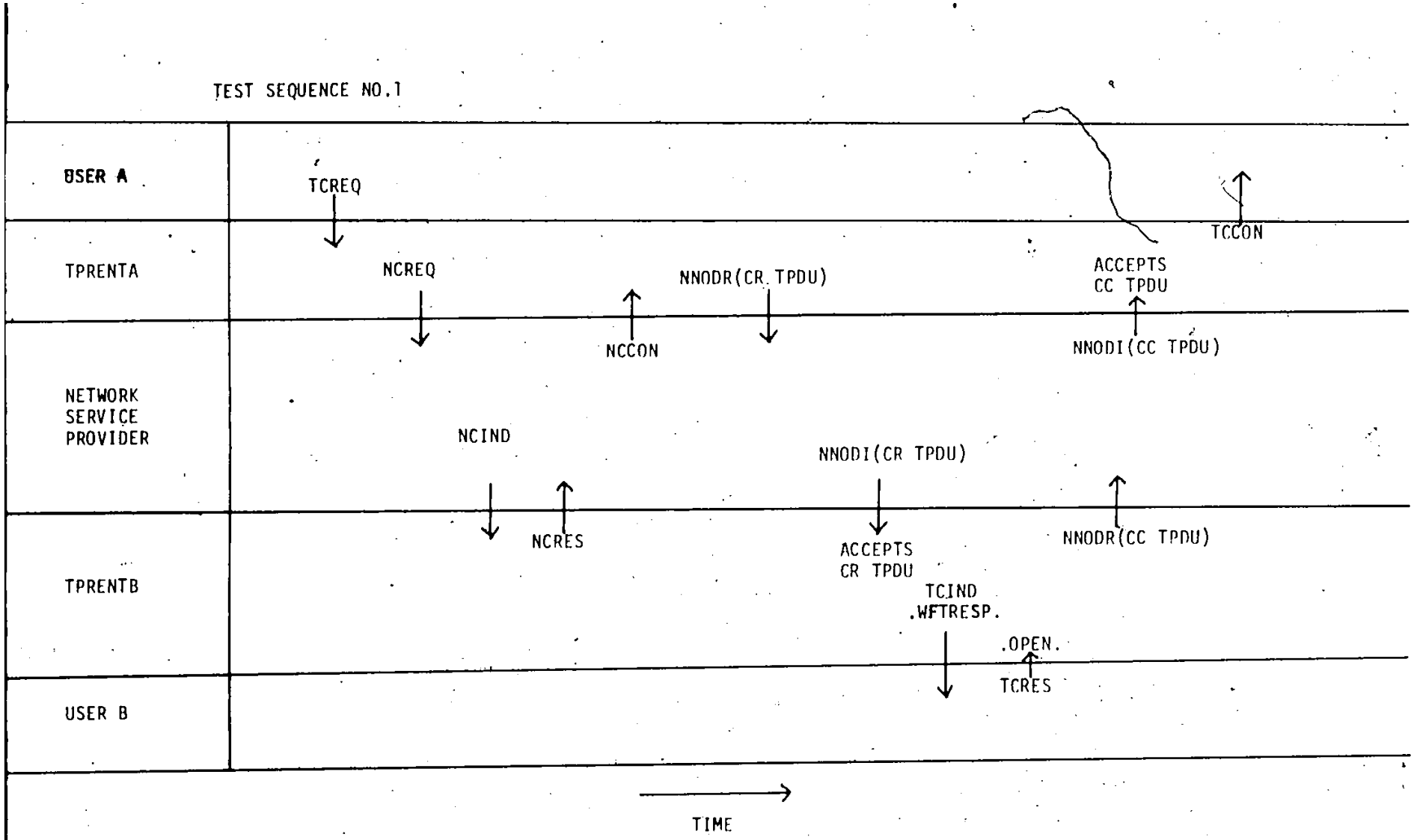


FIG.4.4

SUCCESSFUL TRANSPORT CONNECTION ESTABLISHMENT

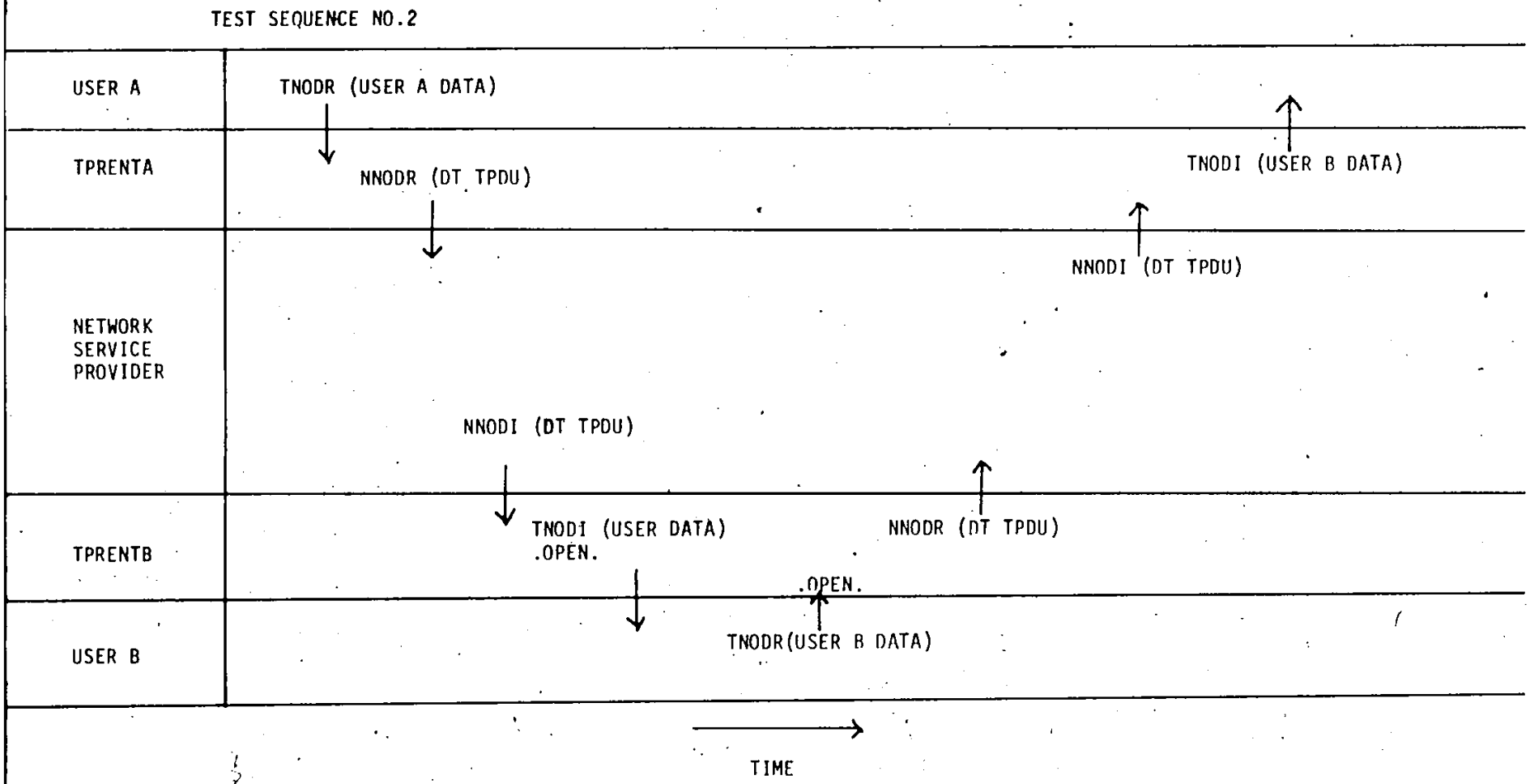


FIG. 4.5

SUCCESSFUL TRANSPORT DATA TRANSFER
(ASSUMING A TRANSPORT CONNECTION HAS BEEN ESTABLISHED)

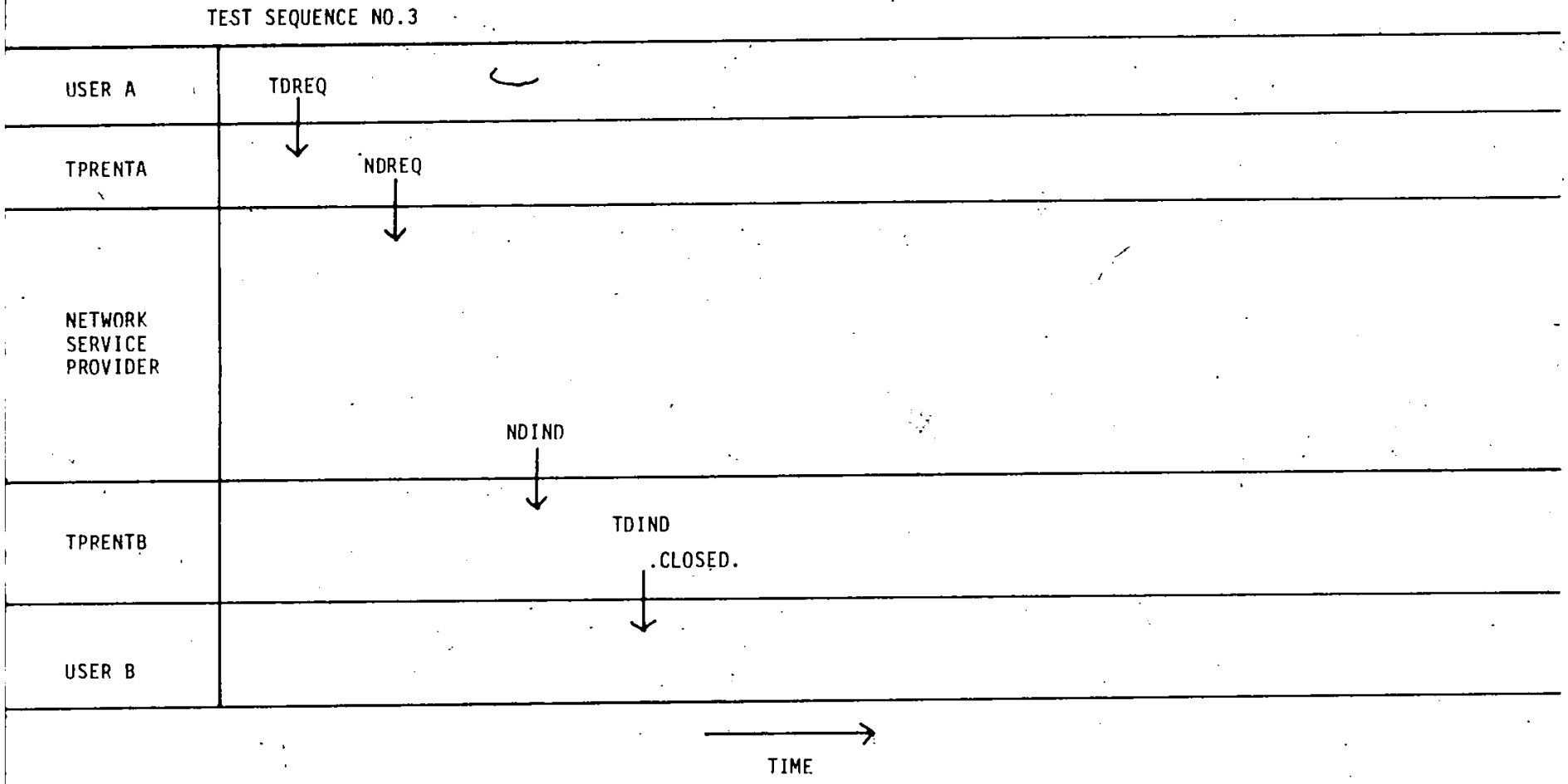


FIG. 4.6

DISCONNECTION OF A TRANSPORT CONNECTION
(ASSUMING A TRANSPORT CONNECTION HAS BEEN ESTABLISHED)

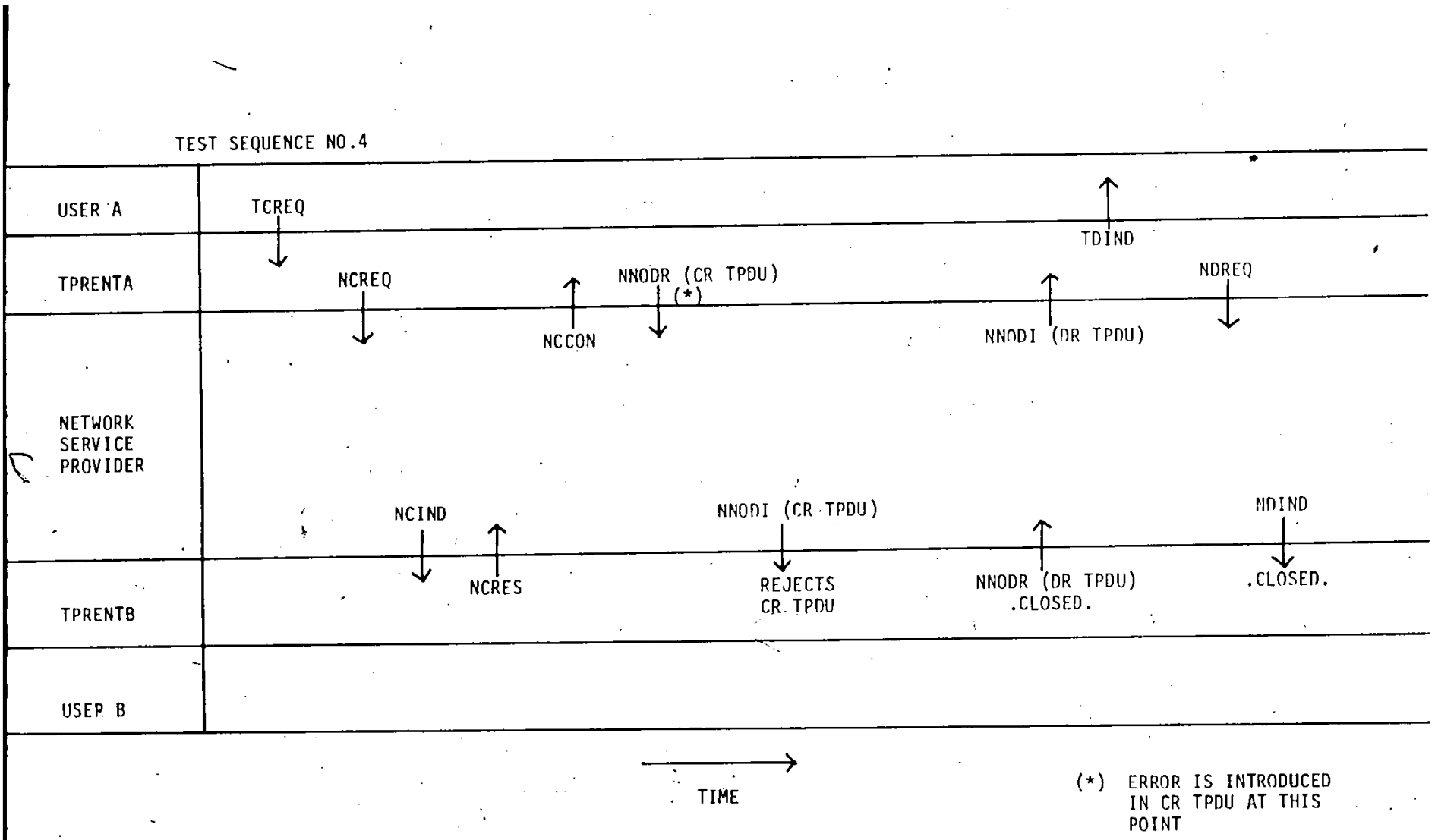


FIG. 4.7

ERRONEOUS CR TPDU SENT TO TPRENTB

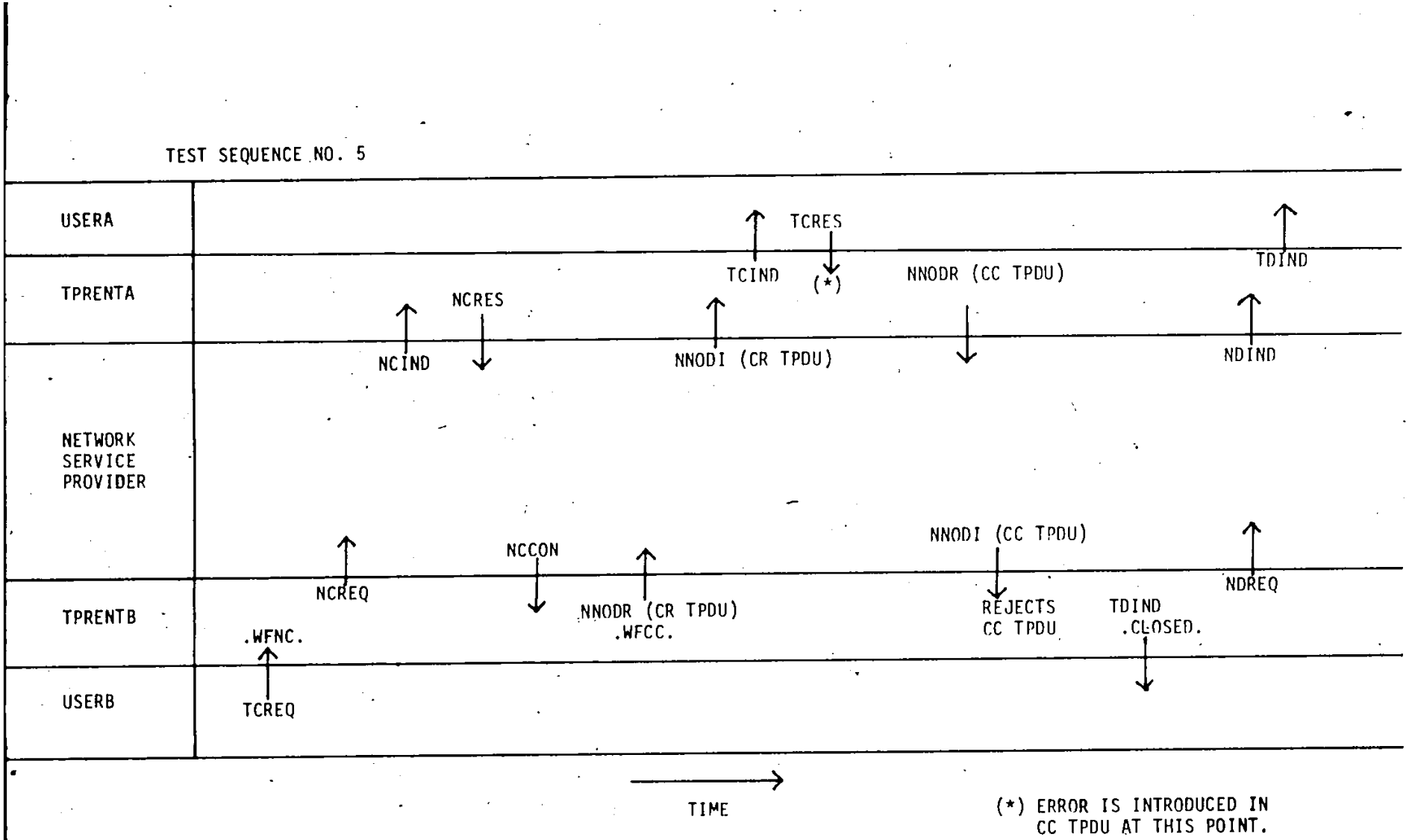


FIG. 4.8

ERRONEOUS CC TPDU SENT TO TPRENTB

TEST SEQUENCE NO.6

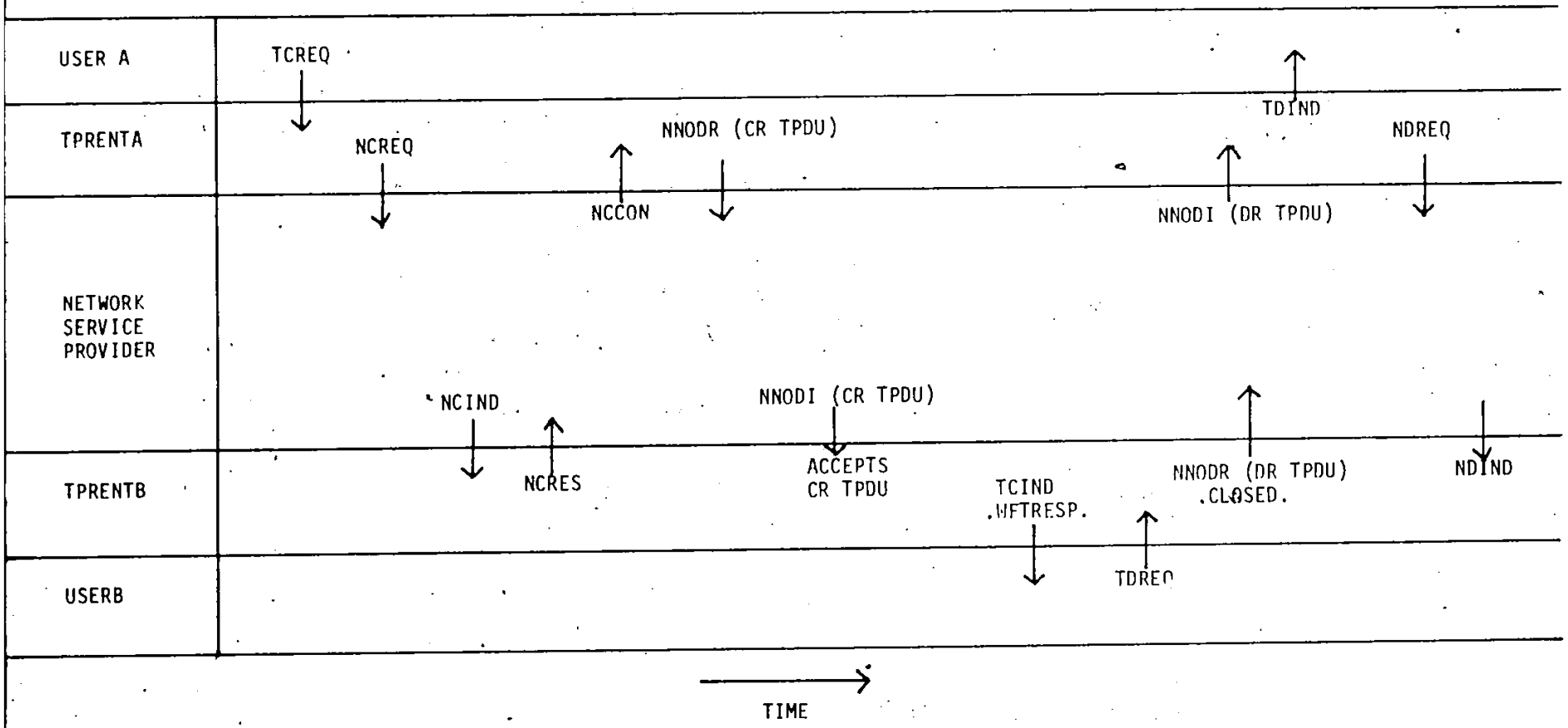


FIG. 4.9

USER A REQUESTS A TRANSPORT CONNECTION BUT USER B REJECTS THE REQUEST

To summarize the experience with running the test sequences, the author feels that the the prototype is easy to use for running the test sequences. The results of these sequences can be readily observed on task TPRENTB's terminal and tabulated. The messages to the tester are quite user friendly. Preparation of the test sequences involves some time and effort if done manually but this process would soon be automated due to a project on automatic test sequence generation using PROLOG conducted by Dr. R. L Probert and Mr. H. Ural of the Computer Science Department, University of Ottawa. All test sequences involving the basic functions of the Transport Protocol Entities i.e Transport Connection Establishment, Transport Data Transfer and Transport Disconnection succeeded. Those related to TPDU error detection and reporting also succeeded. It is to be noted here that two way simultaneous data transfer is possible using the prototypes for TP specifications for as long as the two users desire. Only representative test sequences for the basic functions are presented in the Figs 4.4 through 4.9.

Chapter 5

A SUMMARY, CONCLUSIONS AND SUGGESTIONS FOR FURTHER RESEARCH

The concept of layered network architecture was presented. ISO Reference Model for Open System Interconnection was briefly presented. An introduction to basic concepts of fast prototyping was presented. The existing techniques for protocol testing were summarized. The TS specifications were presented briefly and were implemented in the form of a fast prototype in Pascal. Class-0 TP specifications were briefly presented and were implemented as a fast prototype in Pascal. Experience gained in developing the prototype for TS specifications was utilized in building the Network Service Provider underlying the Transport Protocol Entities. Impact of the layered architecture of ISO OSI Reference Model can be seen in the implementation scheme of the prototype for Class-0 TP specifications. A brief survey of the models for protocol testing architectures was presented. Usefulness of the prototype for TP specifications was shown in an elementary configuration for laboratory testing of communications protocols. This was done by applying test sequences to the prototype to validate the communications requirements of the Class-0 TP specifications.

A Prototype of a set of specifications is faster and more economical to develop compared to the real system. In some cases the ratio of time involved in building a prototype for a set of specifications versus building the real system has been 1:21, [19]. In addition to the time savings involved, the user can reasonably ascertain the correctness of his/her specifications by performing tests on the prototype before actually building the real system. It took us approximately 600 man hours to build the prototype for Class-0 TP specifications whereas in the authors view it would take at least four times the length of time to develop a full blown implementation of the specifications. Degree of detail in building prototypes should be carefully controlled because the essence of building a prototype is to test the basic functions of a future real system. If too much detail is put in the design of a prototype, then the purpose of building a prototype would be defeated and the result would be a real system. The cost-effective use of prototyping techniques in the development and validation of communications protocols should be further investigated.

It was shown in Chapter 4 that realistic experimentation can be performed on a prototype to validate the communication requirements of the TP specifications. The main advantage of building a prototype in a popular high level programming language (e.g Pascal) is that most systems support the language and portability of the prototype would be enhanced.

Following are some possible extensions of the use of the prototype for Class-0 TP specifications developed in this exercise :

1. Although only the Transport layer was considered in this exercise, a similar prototyping approach may be applied to the development and validation of any layer.
2. The prototype of TP specifications can be improved incrementally to be used in standard architectures for laboratory testing of protocols. Task TPRENTA (Transport Protocol Entity A) can be augmented to generate erroneous TPDU's semi-automatically. This can be done by storing the erroneous parameter codes and values in a table. The tester can specify the index to the erroneous value and the TPDU type e.g CR TPDU and the augmented TPRENTA can introduce this erroneous value in the TPDU.
3. The prototype of TP specifications can be enhanced to be used in standard architectures for remote testing of protocols by using an existing network such as DATAPAC. However, substantial low-level modifications will be necessary to the underlying event flag mechanism.

Chapter 6

REFERENCES

[1] Tanenbaum A.S 'Network protocols', ACM Computing Surveys Dec.1981, pp. 453-489.

[2] Green Paul E. Jr. 'An Introduction to Network Architectures and Protocols' IEEE Transactions on Communications Apr.1980, pp. 413-424.

[3] Zimmermann H 'OSI Reference Model--- The ISO Model of Architecture for Open System Interconnection' IEEE Transactions on Communications Apr.1980, pp. 425-432.

[4] Sum Lai Wai 'Protocol Traps in Computer Networks - A Catalog' Bell Northern Research Ltd., May 1982.

[5] Belsnes and Lynning 'Some Problems with the X.25 Packet Level Protocol' ACM Computer Communications Review Vol.7,No.4, 1977, pp. 41-51.

[6] Bochmann G.V 'Notes on the X.25 Procedures For Virtual Call Establishment and Clearing' ACM Computer Communications Review Vol.7,No.4, 1977, pp. 53-59.

[7] West C.H, Zafiropulo 'Automated Validation of a Communications Protocol : the CCITT X.21 Recommendation' IBM Research Development, Vol.22, No.1,Jan.1978, pp. 60-71.

[8] Brand D. and Joyner W.H, Jr. 'Verification of Protocols using Symbolic Execution' IBM Research Report Sept. 1977

[9] Sunshine C.A 'Survey of Communication Protocol Verification Techniques' Symposium on Computer Networks : Trends and Applications' National Bureau of Standards, 1976

[10] Software Reliability Research Group, Univ. of Ottawa 'A study to determine the techniques and tools for Validation of Communications Protocols' Report, Feb. 1982

[11] Bochmann G.V 'Finite State Description of Communications Protocols' Computer Networks North-Holland Publishing Company, 1978, pp.361-372.

[12] Zafiropulo P. 'Protocol Validation by Duologue-Matrix Analysis' IEEE Transactions on Communications Aug.1978, pp. 1187-1194.

[13] Davies D.W, Barber D.L.A, Price W.L, Solomonide 'Communications Networks and their Protocols' John Wiley & Sons, 1979.

[14] Schwabe, Daniel 'Formal Specification and Verification of a Connection Establishment Protocol' IEEE 1981.

[15] desJardins 'Overview and Status of the ISO Reference Model for Open Systems Interconnection', Computer Networks, North-Holland Publishing Company 1981.

[16] Rybczinski A. 'X.25 Interface End-to-End Virtual Circuit Service Characteristics', IEEE Transactions on Communications Apr.1980, pp. 500-510.

[17] Bochmann G.V 'Formal Methods in Communications Protocol Design' IEEE Transactions on Communications Apr.1980, pp. 624-631.

[18] West C.H. 'A General Technique for Communications Protocol Validation' IBM Journal of Research and Development, Jul.1978

[19] Sommervile I, 'Software Engineering', Addison-Wesley Publishing Co. 1982.

[20] Ural H, Probert R. L 'Architectures for Testing Protocol Implementations' Proceedings of CIPS Conference, May 1983, pp. 130-136.

[21] NPL Standards Group 'A System for Testing Protocol Implementations' NPL Report DITC 9/82 Aug. 1982.

[22] Logrippo L 'Specification of Transport Service using Finite-State Transducers and Abstract Data Types'

[23] ISO TC 97/SC 16/WG 5 'Draft Connection Oriented Transport Service Definition', Jan. 1982.

[24] ISO TC 97/SC 16N 1433/WG 6 N74 'Information Processing Systems, Open System Interconnection, Connection Oriented Transport Protocol Specifications', Apr. 1983.

[25] Tanenbaum A. S 'Computer Networks', Prentice Hall Inc. 1981.

[26] Weir D. F, Holmblad J. B, Rothberg A. C 'An X.75 Based Network Architecture' GTE Telenet Communications Corp. USA.

[27] Myers G. J 'The Art of Software Testing', John Wiley & Sons, 1978.

[28] Miller E, Howden W. E 'Tutorial : Software Testing & Validation Techniques' IEEE Computer Society, IEEE Catalog No. EHO 138-8.

[29] ISO /TC 97/SC 16/WG1 'A FDT based on an Extended Transition Model', Nov. 1982.

[30] Carter W. C, Ellozy H. A, Joyner W. H, Leman G. B 'Techniques for Microprogram Validation', IBM Research Report 1977.

[31] Probert R. L, Loisel Lloyd R. R 'CEMS - A Continuous User Feedback Monitoring System for Interactive Applications Software Design and Development', Automated Tools for Information Systems Design, North-Holland Publishing Company, 1982.

[32] Digital Equipment Corporation 'RSX-11M V3.2 Executive Reference Manual'

[33] Software Reliability Research Group, University of Ottawa 'Protocol Testing---Final Report', Feb. 1983.

[34] Bochman G. V, Cerny E, Maksud M, Sarikaya B 'Testing Transport Protocol Implementation', Proceedings of CIPS Conference, May 1983, pp. 123-129.

APPENDIX A

DESCRIPTION OF ROUTINES
OF THE PROTOTYPE FOR TS
SPECIFICATIONS
(User A's side only)

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         BUFFCLEAR
    5
    6      FUNCTION:     To clear the event flag corresponding to the passed integer value.
    7
    8      USAGE:        procedure buffclear(efn : efnum);
    9                                 :
   10                                 :
   11                                 :
   12                    buffclear(efn);
   13
   14      INPUT PARAMETERS:   efn              - the event flag number.
   15
   16      OUTPUT PARAMETERS:  NONE
   17
   18      MODULES CALLED:
   19                          >halt
   20                          >writeln
   21                          clref
   22
   23      EXIT CONDITIONS:
   24
   25                          If the retcode from the CLEF$ Directive is less
   26                          than zero then an error message is printed and
   27                          the event flag may not have been cleared and
   28                          the processor is halted using the 'halt' primitive.
   29
   30      REMARKS:
   31                          See RSX-11M V3.2 Executive Directives Manual on page 6-26 for
   32                          more information.
   33
   34  *****************************************************************************)
```

```
(*$m-*)
(*****************************************************************************

   NAME:        BUFFGO

   FUNCTION:    To set the event flag corresponding to the passed integer value.

   USAGE:       procedure buffgo(efn : efnm);
                        .
                        .
                        .
                buffgo(efn);

   INPUT PARAMETERS:    efn             - the event flag number.

   OUTPUT PARAMETERS:   NONE

   MODULES CALLED:
                        >halt
                        >writeln
                        setef

   EXIT CONDITIONS:     Normal exit conditions.
                        If the retcode from the SETEF$ Directive is less
                        than zero then an error message is printed and the
                        event flag may not be set and the processor is halted
                        using the 'halt' primitive.

   REMARKS:
                See RSX-11M V3.2 Executive Directives Manual on page 6-123 for
                more information.

                Also see chapter 3 of the Manual for more info. on event flags and
                their use.

*****************************************************************************)
```

```
(*$m-*)
(*****************************************************************************

   NAME:          BUFFWAIT

   FUNCTION:      To wait until the event flag corresponding to the passed integer value
                  is set.

   USAGE:         procedure buffwait(efn : efnn);

                  buffwait(efn);

   INPUT PARAMETERS:    efn          : - the event flag number.

   OUTPUT PARAMETERS:   NONE

   MODULES CALLED:
                        >halt
                        >writeln
                        waitfr

   EXIT CONDITIONS:     Normal exit conditions.

                        if the retcode from the WTSE$ Directive is less
                        than zero then an error message is printed and
                        the processor is halted using the 'halt' primitive.

   REMARKS:
                  See RSX-11M V3.2 Executive Directives Manual on page 6-168 for more
                  information.

                  One should be aware of the different types of event flags that
                  can be used for different purposes since for example if global
                  event flags are used for controlling events in a particular task
                  then one must make sure that these flags are not used somewhere
                  else in the system (see charter 3 of Manual).

*****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4     NAME:       READTERM
    5
    6     FUNCTION:   To read in a string from standard input and record its length.
    7
    8     USAGE:      procedure  request(var buffer : request ; var len : integer);extern;
    9                       .
   10                       .
   11                       .
   12                  readterm(buffer, len);
   13
   14     INPUT PARAMETERS:   NONE
   15
   16     OUTPUT PARAMETERS:  buffer          - the string buffer.
   17                         len             - the length of the string buffer.
   18
   19     MODULES CALLED:
   20                         >break
   21                         >eoln
   22                         >read
   23                         >readln
   24                         >writeln
   25
   26     EXIT CONDITIONS:    Normal condition.
   27
   28     REMARKS:
   29                  The inputed line is truncated to REQLEN, so care should
   30                  be taken not to exceed this value.
   31
   32                  It is not recommended to input non-printable characters using
   33                  this module since unpredictable behavior can result when these
   34                  strings are outputed.
   35
   36  *****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4     NAME:         RECA
    5
    6     FUNCTION:     To receive some information sent to the TRNSDA task by using
    7                   the receive data RECEIV$ directive and to return the status
    8                   of the call.
    9
   10    USAGE:         procedure reca(var rbuf : rbuff;
   11                                  var ids : idss); extern;
   12                                  .
   13                                  .
   14                                  .
   15                  reca(rbuf, ids);
   16
   17
   18     INPUT PARAMETERS     NONE
   19
   20     OUTPUT PARAMETERS:   rbuf    - the returned buffer
   21                          ids     - the returned status of the directive call.
   22
   23     MODULES CALLED:      (The symbol '>' designates a primitive)
   24
   25                          >ord
   26                          >writeln
   27                          receiv
   28
   29     EXIT CONDITIONS:
   30
   31                          The return code of RECEIV$  is set to greater or equal
   32                          to 0(zero) in case of successful completion and to less
   33                          than 0 in case of unsuccessful completion.
   34
   35                          For more information on the use of this directive
   36                          and its return codes, see the RSX-11M V3.2 Executive
   37                          Reference Manual.
   38
   39     REMARKS:     NONE
   40
   41  *****************************************************************************)
```

```
(*****************************************************************************

   NAME:        TERMA (MAIN)

   FUNCTION:    This module is used to continuously READ information from
                USER A (TERMINAL A) and to send this information to the TRNSDA
                task.  This information is obtained from terminal input, in the
                form of user primitive strings.

   USAGE:       from MCR : 'RUN TERMA /TASK-TERMA'

   INPUT PARAMETERS:    NONE

   OUTPUT PARAMETERS:   NONE

   MODULES CALLED:      (The symbol '>' designates a primitive)

                        >halt
                        >read
                        >writeln
                        buffclear
                        buffgo
                        buffwait
                        readterm
                        send

   EXIT CONDITIONS:
                        This program has been designed to run indefinitely until
                        the task into which it runs is specifically aborted with
                        the MCR command "ABORT TERMA".

                        If any errors are detected by one of the routines called,
                        then the routine itself will print an error message and
                        halt the processor.  The exception to this is the call
                        to the SDAT$ ('send') routine.  In this case TERMA checks
                        the return status and in case of error, prints out an
                        error message and then halts the processor by using the
                        'halt' primitive.

   REMARKS:     NONE

*****************************************************************************)
```

```
    1   (*****************************************************************************
    2
    3      NAME:         TRNSDA (MAIN)
    4
    5      FUNCTION:     Transducer for side 'A'.
    6
    7      USAGE:        from MCR : 'RUN TRNSDA /TASK=TRNSDA'
    8
    9     INPUT PARAMETERS:    NONE
   10
   11      OUTPUT PARAMETERS:   NONE
   12
   13      MODULES CALLED:      (The symbol '>' designates a primitive)
   14
   15                           >halt
   16                           >write
   17                           >writeln
   18                           buffclear
   19                           buffgo
   20                           buffwait
   21                           mark
   22                           reca
   23                           waab
   24                           wtermm
   25
   26      EXIT CONDITIONS:
   27                           This program has been designed to run indefinitely until
   28                           the task into which it runs is specifically aborted with
   29                           the MCR command 'ABORT TRNSDA'.
   30
   31                           If any errors are detected by one of the routines called,
   32                           then the routine itself will print an error message and
   33                           halt the processor by using the 'halt' primitive.  The
   34                           exception to this is the RECA routine.  TRNSDA checks
   35                           the status of the call to RECA and if there is an error
   36                           condition, then an error message is printed and the
   37                           processor is halted by TRNSDA.
   38
   39      REMARKS:
   40                           See Ref. (22) for more details.
   41
   42   *****************************************************************************)
```

```
 1  (*$m-*)
 2  (*****************************************************************************
 3
 4     NAME:        WQAB
 5
 6     FUNCTION:    To put a symbol into the A-->B queue.
 7
 8     USAGE:       procedure wqab(messageXq : transQXresp); extern;
 9                                .
10                                .
11                                .
12                  wqab(messageXq);
13
14     INPUT PARAMETERS:    messageXq  - the symbol to be sent to the queue.
15
16     OUTPUT PARAMETERS:   NONE
17
18     MODULES CALLED:      (The symbol '>' designates a primitive)
19
20                          >halt
21                          >writeln
22                          send
23
24     EXIT CONDITIONS:
25                          If there are any errors detected in calling SDAT$('send')
26                          then an error message identifying the error and
27                          error code is printed onto standard I/O and the processor
28                          is then halted by using the 'halt' primitive.
29
30                          The return code of SDAT$ is set to greater or equal to
31                          0 (zero) in case of successful completion and less than
32                          0 in case of unsuccessful completion.
33
34                          For more information on the use of these system routines
35                          and their return codes, see the RSX-11M V3.2 Executive
36                          Reference Manual.
37
38     REMARKS:     NONE
39
40  *****************************************************************************)
```

```
PASCAL  PDP-11  VERSION  6.07        1983-08-06      13:03:13          PAGE   1
WTERMA.PAS                              .MAIN.

    1   (*$m-*)
    2   (*************************************************************************** *
    3
    4      NAME:         WTERMA
    5
    6      FUNCTION:     To write a TRNSDA error message on the terminal in case
    7                    illegal requests are made by TRNSDB or terminal A or to
    8                    write a query status message or one of the following
    9                    messages (successful): tccom, tcmd, tdind, tmodi, tendi.
   10
   11     USAGE:        procedure wterma(messno : integer;
   12                                    stateXa : state;
   13                                    rreq : request); extern;
   14
   15     INPUT PARAMETERS:    messno  - the type of message.
   16                          stateXa - the state of TRNSDA.
   17                          rreq    - the inputed string.
   18
   19     OUTPUT PARAMETERS:   NONE
   20
   21     MODULES CALLED:      (The symbol '>' designates a primitive)
   22
   23                          >halt
   24                          >writeln
   25
   26     EXIT CONDITIONS:
   27                          Normal exit conditions.
   28
   29     REMARKS:      NONE
   30
   31  ***************************************************************************)
```

APPENDIX B

PASCAL CODE OF ROUTINES
OF THE PROTOTYPE FOR TS
SPECIFICATIONS
(User A's side only)

```
1   (*$m-*)
2   (*****************************************************************************
3
4      NAME:          BUFFCLEAR
5
6      FUNCTION:      To clear the event flag corresponding to the passed integer value.
7
8      USAGE:         procedure buffclear(efn : efnn);
9                                 .
10                                .
11                                .
12                     buffclear(efn);
13
14     INPUT PARAMETERS:    efn              - the event flag number.
15
16     OUTPUT PARAMETERS:   NONE
17
18     MODULES CALLED:
19                          >halt
20                          >writeln
21                          clref
22
23     EXIT CONDITIONS:
24
25                          If the retcode from the CLEF$ Directive is less
26                          than zero then an error message is printed and
27                          the event flag may not have been cleared and
28                          the processor is halted using the 'halt' primitive.
29
30     REMARKS:
31                          See RSX-11M V3.2 Executive Directives Manual on page 6-26 for
32                          more information.
33
34  *****************************************************************************)
35
36  type
37          efnn = integer;
38
39  procedure buffclear(efn : efnn);
40
41  var  ids :integer;
42
43  procedure clref(var efn : integer; var ids : integer); extern(fortran);
```

```
PASCAL  PDP-11  VERSION  6.07    1908-08-04      03:23:55          PAGE    2
BUFFCLEAR.PAS                   BUFFCLEAR

    44
    45   begin
    46   { writeln(tty, 'entering buffclear routine'));}
    47
    48     clref(efn, ids);    (* call CLEF$ *)
    49     if ids  0 then              (* check if retcode is less than 0 *)
    50     begin
    51       writeln( error in doing clref. Error code -', ids);
    52       halt
    53     end
    54
    55   { writeln(tty, 'leaving buffclear routine'));
    56   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      000256
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 6 SECONDS

PAS BUFFCLEAR+BUFFCLEAR/P42-BUFFCLEAR
```

```
 1  (*$m-*)
 2  (*****************************************************************************
 3
 4     NAME:         BUFFGO
 5
 6     FUNCTION:     To set the event flag corresponding to the passed integer value.
 7
 8     USAGE:        procedure buffgo(efn : efnn);
 9
10
11
12                   buffgo(efn);
13
14     INPUT PARAMETERS:    efn              - the event flag number.
15
16     OUTPUT PARAMETERS:   NONE
17
18     MODULES CALLED:
19                          >halt
20                          >writeln
21                          setef
22
23     EXIT CONDITIONS:     Normal exit conditions.
24                          If the retcode from the SETEF$ Directive is less
25                          than zero then an error message is printed and the
26                          event flag may not be set and the processor is halted
27                          using the 'halt' primitive.
28
29     REMARKS:
30                   See RSX-11M V3.2 Executive Directives Manual on page 6-123 for
31                   more information.
32
33                   Also see chapter 3 of the Manual for more info. on event flags and
34                   their use.
35
36  *****************************************************************************)
37
38  type
39          efnn = integer;
40
41  procedure buffgo(efn : efnn);
42
43  var  ids :integer;
```

```
   44
   45  procedure setef(var efn:integer; var ids : integer); extern(fortran);
   46
   47  begin
   48  { writeln(tty, 'entering buffgo routine');}
   49
   50    setef(efn, ids);      (* call the SETF$ Directive *)
   51    if ids < 0 then               (* check if retcode is less than 0 *)
   52    begin
   53      writeln(' error in setef. Error code - ', ids);
   54      halt
   55    end
   56
   57  { writeln(tty, 'leaving buffgo routine');}
   58  end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000250
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 5 SECONDS

PAS BUFFGO,BUFFGO/P42-BUFFGO
```

```
 1  (*$m-*)
 2  (*****************************************************************************
 3
 4     NAME:          BUFFWAIT
 5
 6     FUNCTION:      To wait until the event flag corresponding to the passed integer value
 7                    is set.
 8
 9     USAGE:         procedure buffwait(efn : efnm);
10
11
12
13                    buffwait(efn);
14
15     INPUT PARAMETERS:    efn              - the event flag number.
16
17     OUTPUT PARAMETERS:   NONE
18
19     MODULES CALLED:
20                          lib$halt
21                          writeln
22                          warlff
23
24     EXIT CONDITIONS:     Normal exit conditions.
25
26                          if the retcode from the WTSE$ Directive is less
27                          than zero then an error message is printed and
28                          the processor is halted using the 'lib$halt' primitive.
29
30     REMARKS:
31                    See RSX-11M V3.2 Executive Directives Manual on page 6-168 for more
32                    information.
33
34                    One should be aware of the different types of event flags that
35                    can be used for different purposes since for example if global
36                    event flags are used for controlling events in a particular task
37                    then one must make sure that these flags are not used somewhere
38                    else in the system (see chapter 3 of manual).
39
40 *****************************************************************************)
41
42 type
43       efnm = integer;
```

```
   44
   45   procedure buffwait(efn : efnn);
   46
   47   var ids : integer;
   48
   49   procedure waitfr(var efn:integer; var ids : integer); extern(fortran);
   50
   51   begin
   52   { writeln(tty, 'entering buffwait routine ')}
   53
   54     waitfr(efn, ids);     (* wait for flag to be set *)
   55
   56     if ids <> 0 then               (* was it successful ? *)
   57     begin (* no *)
   58       writeln(' error in waitfr. Error code = ', ids);
   59       halt
   60     end
   61
   62   { writeln(tty, 'leaving buffwait routine ')}
   63   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000252
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 7 SECONDS

PAS BUFFWAIT,BUFFWAIT/P42-BUFFWAIT
```

```
    1   ($$m-$)
    2   (*****************************************************************************
    3
    4      NAME:         READTERM
    5
    6      FUNCTION:     To read in a string from standard input and record its length.
    7
    8      USAGE:        procedure  request(var buffer : request ; var len : integer);extern;
    9                                .
   10                                .
   11                                .
   12                    readterm(buffer, len);
   13
   14      INPUT PARAMETERS:    NONE
   15
   16      OUTPUT PARAMETERS:   buffer          - the string buffer.
   17                           len             - the length of the string buffer.
   18
   19      MODULES CALLED:
   20                           .break
   21                           .eoln
   22                           .read
   23                           .readln
   24                           .writeln
   25
   26      EXIT CONDITIONS:     Normal condition.
   27
   28      REMARKS:
   29                    The inputed line is truncated to REQLEN, so care should
   30                    be taken not to exceed this value.
   31
   32                    It is not recommended to input non-printable characters using
   33                    this module since unpredictable behavior can result when these
   34                    strings are outputed.
   35
   36  *****************************************************************************)
   37
   38  const                                          ( GLOBAL CONSTANT )
   39          REQLEN = 6;
   40  type                                           ( GLOBAL TYPE )
   41          request = array [1..REQLEN] of char;
   42
   43  procedure  readterm(var buffer : request ; var len : integer);
```

```
   44
   45   const                                    { LOCAL VT100 TERMINAL CONSTANTS }
   46           revvideo = '[7m';
   47           video    = '[0m';
   48           up       = '[A';
   49
   50   var     i : integer;
   51
   52   begin
   53           (* set input into reverse video *)
   54           writeln(CHR(27), revideo, CHR(27), up);
   55
   56           (* read the input line and record its length *)
   57           break(tty);
   58           readln(tty);
   59           i := 0;
   60           if not(eoln(ttyin)) then
   61           begin
   62                   i := 1;
   63                   (* length of string must not exceed REQLEN *)
   64                   while (not (eoln(ttyin)) and (i <= REQLEN)) do
   65                   begin
   66                           read(tty, buffer[i]);
   67                           i := i + 1
   68                   end;
   69                   len := i - 1;
   70           end
   71           else    (* empty line *)
   72                   len := 1;
   73
   74           (* put back in normal display mode *)
   75           writeln(CHR(27), video, CHR(27), up)
   76   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000840
OUTERHOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 11 SECONDS

PAS READTERM,READTERM/P42-READTERM
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         RECA
    5
    6      FUNCTION:     To receive some information sent to the TRNSDA task by using
    7                    the receive data RECEIV$ directive and to return the status
    8                    of the call.
    9
   10     USAGE:        procedure reca(var rbuf : rbuff;
   11                                  var ids : idss); extern;
   12                          .
   13                          .
   14                          .
   15                   reca(rbuf, ids);
   16
   17
   18      INPUT PARAMETERS     NONE
   19
   20      OUTPUT PARAMETERS:   rbuf    - the returned buffer
   21                           ids     - the returned status of the directive call.
   22
   23      MODULES CALLED:      (The symbol '>' designates a primitive)
   24
   25                           >ord
   26                           >writeln
   27                           receiv
   28
   29      EXIT CONDITIONS:
   30
   31                           The return code of RECEIV$  is set to greater or equal
   32                           to 0(zero) in case of successful completion and to less
   33                           than 0 in case of unsuccessful completion.
   34
   35                           For more information on the use of this directive
   36                           and its return codes, see the RSX-11M V3.2 Executive
   37                           Reference Manual.
   38
   39      REMARKS:     NONE
   40
   41  *****************************************************************************)
   42
   43  const
```

```
   44          REQUESTLEN = 6;
   45          BUFLEN     = 13;
   46          DUMLEN     = 8;
   47
   48  type
   49
   50          transdXresp = (a, b, c, d, i, e, x);
   51
   52          request = packed array[1..REQUESTLEN] of char;
   53
   54          tskk = record
   55                   nam : array[1..2] of integer;
   56          end;
   57
   58          rbuff =  record
   59                   nam : tskk;
   60                   req : request;
   61                   typ : boolean;
   62                   ch  : transdXresp;
   63                   dum : array[1..DUMLEN] of integer;
   64          end;
   65
   66          efnn = integer;
   67
   68          idss = integer;
   69
   70  procedure reca(var rbuf : rbuff;
   71                 var ids : idss);
   72
   73  var
   74        dummy   : efnn;
   75        tsk     : tskk;
   76
   77  procedure receiv(var tsk : tskk;
   78                   var rbuf : rbuff;
   79                   var dummy : efnn;
   80                   var ids : idss); extern(fortran);
   81
   82  begin
   83  { writeln(tty, ' entering reca procedure');}
   84
   85    receiv(tsk, rbuf, dummy, ids);
   86
```

```
PASCAL  PDP-11  VERSION  6.07      1983-08-06      12:35:33          PAGE   3
RECA.PAS                                RECA

   87    { writeln(tty, 'user request - ', rbuf.req, ' use char - ', ord(rbuf.ch))}
   88
   89    { writeln(tty, ' leaving reca procedure')}
   90    end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE        000106
OUTERMOST DATA SIZE       000002
RESERVED STACK & HEAP     000526
COMPILATION TIME 6 SECONDS

RUN RECA,RECA/P42-RECA
```

```
    1   (*****************************************************************************
    2
    3       NAME:       TERMA (MAIN)
    4
    5       FUNCTION:   This module is used to continuously READ information from
    6                   USER A (TERMINAL A) and to send this information to the TRNSDA
    7                   task.  This information is obtained from terminal input, in the
    8                   form of user primitive strings.
    9
   10       USAGE:      from MCR : 'RUN TERMA /TASK=TERMA'
   11
   12       INPUT PARAMETERS:   NONE
   13
   14       OUTPUT PARAMETERS:  NONE
   15
   16       MODULES CALLED:     (The symbol ' ' designates a primitive)
   17
   18                           'halt
   19                           'read
   20                           'writeln
   21                           buffclear
   22                           boffgo
   23                           buffwait
   24                           readterm
   25                           send
   26
   27       EXIT CONDITIONS:
   28                           This program has been designed to run indefinitely until
   29                           the task into which it runs is specifically aborted with
   30                           the MCR command 'ABORT TERMA'.
   31
   32                           If any errors are detected by one of the routines called,
   33                           then the routine itself will print an error message and
   34                           halt the processor.  The exception to this is the call
   35                           to the SDAT$ ('send') routine.  In this case TERMA checks
   36                           the return status and in case of error prints out an
   37                           error message and then halts the processor by using the
   38                           'halt' primitive.
   39
   40       REMARKS:    NONE
   41
   42   *****************************************************************************)
   43
```

```
   44  program terma(tty);
   45
   46  const
   47          TERMAFLAG  = 42;
   48          TRNSDAFLAG = 43;
   49          REQUESTLEN = 6;
   50          BUFLEN     = 13;
   51          DUMLEN     = 8;  (* BUFLEN - 2 - (REQUESTLEN + 1) / 2 *)
   52
   53  type
   54          transdXresp = (ur, ar, sr, dr, ir, er, x);
   55
   56          request = packed array[1..REQUESTLEN] of char;
   57
   58          buff =  record
   59                      req : request;
   60                      typ : boolean;
   61                      ch  :transdXresp;
   62                      dum : array[1..DUMLEN] of integer;
   63                  end;
   64
   65          tskk = record
   66                      num : array[1..2] of integer;
   67                  end;
   68
   69          efnn = integer;
   70
   71          idss = integer;
   72
   73  var
   74          buf     : buff;
   75          efn     : efnn;
   76          ids     : idss;
   77          j       : integer;
   78          len     : integer;
   79          tsk     : tskk;
   80
   81  procedure buffclear(efn : efnn); extern;
   82
   83  procedure buffset(efn : efnn); extern;
   84
   85  procedure buffwait(efn : efnn); extern;
   86
```

```
 87  procedure readterm(var req : request;
 88                     var len : integer); extern;
 89
 90  procedure send(var tsk : tskk;
 91                 var buf : buff;
 92                 var efn : efnn;
 93                 var ids : idss); extern(fortran);
 94
 95  begin                                              (* TERMA *)
 96    writeln(tty, ' Task TERMA activated.....');
 97
 98    (* set up the receiver task name 'TRNSDA'*)
 99    tsk.nam[1] := 32734;
100    tsk.nam[2] := 30562;
101
102    (* set up event flag to wake up receiver task *)
103    efn := TRNSDAFLAG;
104
105    (* set up type of transfer (string). *)
106    buf.typ := true;
107
108    (* initialize the rest of the buffer to be sent *)
109    buf.ch := a;
110    buf.dum[1] := 1;
111    buf.dum[2] := 2;
112    buf.dum[3] := 3;
113    buf.dum[4] := 4;
114    buf.dum[5] := 5;
115    buf.dum[6] := 6;
116    buf.dum[7] := 7;
117    buf.dum[8] := 8;
118
119    while (1 = 1)do                (* RUN forever *)
120    begin
121
122      (* put user primitive into the appropriate buffer field *)
123      readterm(buf.req, len);
124
125      (* pad string with blanks if needed *)
126      for j := (len + 1) to REQUESTLEN do
127        buf.req[j] := ' ';
128
129      (* wait until TERMA has priority to send to TRNSDA *)
```

```
 130       buffwait(TERMAFLAG);
 131
 132       (* send data buffer to TRNSDA *)
 133       send(tsk, buf, efn, ids);
 134       if (ids < 0) then
 135       begin
 136         writeln(tty, 'error in calling SEND in TERMA, error code - ', ids);
 137         halt
 138       end;
 139     end;
 140
 141     writeln(tty, ' Task TERMA terminated.....')
 142   end.
```

NO ERROR DETECTED
TOTAL PROGRAM SIZE 001534
OUTERMOST DATA SIZE 000050
RESERVED STACK & HEAP 001100
COMPILATION TIME 15 SECONDS

PAS TERMA,TERMA/P42=TERMA

```
    1  (*****************************************************************************
    2
    3     NAME:         TRNSDA (MAIN)
    4
    5     FUNCTION:     Transducer for side 'A'.
    6
    7     USAGE:        from MCR : 'RUN TRNSDA /TASK=TRNSDA'
    8
    9     INPUT PARAMETERS:    NONE
   10
   11     OUTPUT PARAMETERS:   NONE
   12
   13     MODULES CALLED:      (The symbol '>' designates a Primitive)
   14
   15                          >halt
   16                          >write
   17                          >writeln
   18                          buffclear
   19                          buffso
   20                          buffwait
   21                          mark
   22                          reca
   23                          waab
   24                          wierma
   25
   26     EXIT CONDITIONS:
   27                          This program has been designed to run indefinitely until
   28                          the task into which it runs is specifically aborted with
   29                          the MCR command 'ABORT TRNSDA'.
   30
   31                          If any errors are detected by one of the routines called,
   32                          then the routine itself will print an error message and
   33                          halt the processor by using the 'halt' primitive.  The
   34                          exception to this is the RECA routine.  TRNSDA checks
   35                          the status of the call to RECA and if there is an error
   36                          condition, then an error message is printed and the
   37                          processor is halted by TRNSDA.
   38
   39     REMARKS:
   40                          See Ref. (22) for more details.
   41
   42  *****************************************************************************)
   43
```

```
program trnsda(tty);

const
        TERMBFLAG  = 40;
        TRNSDBFLAG = 41;
        TERMAFLAG  = 42;
        TRNSDAFLAG = 43;
        REQUESTLEN = 6;
        BUFLEN     = 13;
        DUMLEN     = 8; (* BUFLEN - 2 - (REQUESTLEN + 1) / 2 *)

type
        transXresp = (a, 'a, s, d, i, e, x);

        request = packed array[1..REQUESTLEN] of char;

        tskk =  record
                        nam : array[1..2] of integer;
                end;

        rbuff = record
                        nam : tskk;
                        req : request;
                        typ : boolean;
                        ch  : transXresp;
                        dum : array[1..DUMLEN] of integer;
                end;

        buff = record
                        req : request;
                        typ : boolean;
                        ch  : transXresp;
                        dum : array[1..DUMLEN] of integer;
                end;

        state = (one, two, three, four);

        efnn = integer;

        idss = integer;

        tntt = integer;
```

```
 87          tmgg = integer;
 88
 89   var
 90          b           : char;
 91          buf         : buff;
 92          efn         : efnn;
 93          ids         : idss;
 94          rbuf        : rbuff;
 95          stateXa     : state;
 96          tmg         : tmgg;
 97          tnt         : tntt;
 98
 99   procedure buffclear(efn : efnn); extern;
100
101   procedure buffsu(efn : efnn); extern;
102.
103   procedure buffwait(efn : efnn); extern;
104
105   procedure mark(var efn : efnn;
106                  var tmg : tmgg;
107                  var tnt : tntt;
108                  var ids : idss); extern(fortran);
109
110   procedure reca(var rbuf : rbuff;
111                  var ids  : idss); extern;
112
113   procedure wsab(messageXu : transXresp); extern;
114
115   procedure wterma(messageXuser : integer;
116                    stateXa : state;
117                    ddd : request); extern;
118
119   begin                (* TRNSDA *)
120
121     (* initialize state to IDLE state *)
122     stateXa := one;
123
124     writeln(tty, 'TRNSDA is activated.....');
125
126     (* initialize waiting in between empty queues *)
127     tmg := 2;  (* 2       *)
128     tnt := 2;  (* seconds *)
129
```

```
130     while (1 = 1) do              (* DO forever *)
131     begin
132
133       (* wait until TRNSDA has priority *)
134       buffwait(TRNSDAFLAG);
135
136       (* remove priority from TERMA *)
137       buffclear(TERMAFLAG);
138
139       (* receive data from TRNSDA task queue *)
140       recv(rbuf, ids);
141       if (ids = 1) then           (* successful receive *);
142       begin
143
144         if rbuf.typ then          (* message from terminal A *)
145         begin
146
147           case stateXa of         (* select the proper state *)
148
149
150         (* if in idle state then do the following *)
151
152
153     one :  begin
154              if (rbuf.req = 'ncreq ') then
155              begin
156                stateXa  :- two;                (* outXconnXpending  *)
157                wusb(u)
158              end
159              else
160                  wterma(6, stateXa, rbuf.req)
161            end;                                (* idle *)
162
163         (* if in outXconnXpending state then, do the following *)
164
165
166     two :  begin
167              if (rbuf.req = 'ndreq ') then
168              begin
169                stateXa :- one;                 (* back to idle state *)
170                wusb(d)                         (* SEND data to NTRNSDB *)
171              end
172              else
```

```
123                 wterma(6, stateXa, rbuf.req)
174               end;                              (* outXconnXpending *)
175
176
177             (* if in inXconnXpending state, then do the following *)
178
179
180       three : begin
181                 if (rbuf.req = 'idreq ') then
182                 begin
183                   stateXa := one;             (* back to idle *)
184                   waab(d)                     (* SEND data to NTRNSDB *)
185                 end
186               else if (rbuf.req = 'tcres ') then
187                 begin
188                   stateXa := four;            (* dataXtransferXrdy *)
189                   waab(s)
190                 end
191                 else
192                   wterma(6, stateXa, rbuf.req)
193               end;                    (* INXCONNXPENDING *)
194
195
196             (* if in dataXtransferXrdy state, then do the following *)
197
198
199       four : begin
200                 if (rbuf.req = 'tdreq ') then
201                 begin
202                   waab(d);                     (* SEND data to NTRNSDB *)
203                   stateXa := one               (* back to idle *)
204                 end
205                 else if (rbuf.req = 'tnodr ') then
206                 begin
207                   waab(i)
208                 end
209                 else if (rbuf.req = 'texdr ') then
210                 begin
211                   waab(x)
212                 end
213                 else
214                   wterma(6, stateXa, rbuf.req)
215               end                                (* dataXtransferXrdy *)
```

```
216          end                                    (* case *)
217        end                                      (* incoming user message *)
218    (*
219
220
221      the following section deals with the messages coming in from the
222      queueXbXtoXa
223
224    *)
225        else if (rbuf.ch  - e) then
226          wterma(7, stateXa, rbuf.req)
227        else case stateXa of
228
229    (* if in idle state, then do the following *)
230
231      one:begin
232            if (rbuf.ch - a) then
233            begin
234              wterma(2, stateXa, rbuf.req);
235              stateXa := three                   (* inXconnXpending *)
236            end
237            else
238            begin
239              waab(e)
240            end
241          end;                                   (* idle *)
242
243
244        (* if in outXconnXpending state, then do the following     *)
245
246
247      two:begin
248            if (rbuf.ch = d) then
249            begin
250              wterma(3, stateXa, rbuf.req);
251              stateXa := one                     (* back to idle *)
252            end
253            else if (rbuf.ch = s) then
254            begin
255              wterma(1, stateXa, rbuf.req);      (* SEND data NCCON to TPRENTA *)
256              stateXa := four                    (* dataXtransferXrdy *)
257            end
258            else
```

```
259             begin
260               waab(e)                          (* SEND data to NTRNSDB *)
261             end
262           end;                                  (* OUTXCONNXPENDING *)
263
264
265         (*      if in inXconnXpending state, then do the following    *)
266
267       three :
268           begin
269             if (rbuf.ch = d) then
270             begin
271               wterma(3, stateXa, rbuf.req);
272               stateXa := one
273             end
274             else
275             begin
276               waab(e)
277             end
278           end;                                  (* inXconnXpending *)
279
280
281       (*      if in dataXtransferXrdy state, then do the following   *)
282
283   four:   begin
284             if (rbuf.ch = d) then
285             begin
286               wterma(3, stateXa, rbuf.req);
287               stateXa := one                    (* back to idle *)
288             end
289             else if (rbuf.ch = i) then
290             begin
291               wterma(4, stateXa, rbuf.req)
292             end
293             else if (rbuf.ch = x) then
294             begin
295               wterma(5, stateXa, rbuf.req)
296             end
297             else
298             begin
299               waab(e)
300             end
301           end                                   (* dataXtransferXrdy *)
```

```
  302         end                             (* case and message from terminal *)
  303       end                               (* if successful receive *)
  304       else if (ids <>-8) then             (* error in receive *)
  305       begin
  306       writeln(tty, 'error in reading from TRNSDA task queue');
  307         halt
  308       end;
  309
  310       (* give priority to other terminal *)
  311
  312       buffgo(TERMBFLAG);
  313       (* start up TRNSDB after 2 seconds *)
  314       efn := TRNSDBFLAG;
  315       mark(efn, tmg, tnt, ids);
  316
  317       (* reset priority of TRNSDA *)
  318       buffclear(TRNSDAFLAG)
  319
  320
  321     end;                                (* end of RUN forever *)
  322
  323       writeln(tty, 'NTRNSDA TASK TERMINATED.....................')
  324   end.
```

NO ERROR DETECTED
TOTAL PROGRAM SIZE 004530
OUTERHOST DATA SIZE 000106
RESERVED STACK & HEAP 001134
COMPILATION TIME 34 SECONDS

PAS TRNSDA;TRNSDA/P42-TRNSDA

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:        WQAB
    5
    6      FUNCTION:    To put a symbol into the A-B queue.
    7
    8      USAGE:       procedure wqab(messageXa : transdXresp); extern;
    9
   10                                 .
   11                                 .
   12                   wqab(messageXa);
   13
   14      INPUT PARAMETERS:    messageXa  - the symbol to be sent to the queue.
   15
   16      OUTPUT PARAMETERS:   NONE
   17
   18      MODULES CALLED:      (The symbol '>' designates a primitive)
   19
   20                           >halt
   21                           >writeln
   22                           send
   23
   24      EXIT CONDITIONS:
   25                           If there are any errors detected in calling SDAT$('send')
   26                           then an error message identifying the error and
   27                           error code is printed onto standard I/O and the processor
   28                           is then halted by using the 'halt' primitive.
   29
   30                           The return code of SDAT$ is set to greater or equal to
   31                           0 (zero) in case of successful completion and less than
   32                           0 in case of unsuccessful completion.
   33
   34                           For more information on the use of these system routines
   35                           and their return codes, see the RSX-11M V3.2 Executive
   36                           Reference Manual.
   37
   38      REMARKS:     NONE
   39
   40  *****************************************************************************)
   41
   42  const
   43          QABFLAG     = 36;
```

```
44          REQUESTLEN = 6;
45          BUFLEN     = 13;
46          DUMLEN     = 8;  (* BUFLEN - 2 - (REQUESTLEN + 1) / 2 *)
47
48      type
49          transdXresp = (a, b, c, d, e, f, x);
50
51          request = packed array[1..REQUESTLEN] of char;
52
53          buff =  record
54                      req : request;
55                      typ : boolean;
56                      ch  : transdXresp;
57                      dum : array[1..DUMLEN] of integer;
58              end;
59
60          tskk = record
61                      nam : array[1..2] of integer;
62              end;
63
64          efnn = integer;
65
66          idss = integer;
67
68      procedure wqab(message : transdXresp);
69
70      var
71              buf    : buff;
72              efn    : efnn;
73              ids    : idss;
74              len    : integer;
75              tsk    : tskk;
76
77      procedure send(var tsk : tskk;
78                     var buf : buff;
79                     var efn : efnn;
80                     var ids : idss); extern(fortran);
81
82      begin
83
84      { writeln(tty, ' entering wqab procedure'); }
85
86        (* set up task name = 'TRNSDB' *)
```

```
   87     tsk.nam[1] := 32734;
   88     tsk.nam[2] := 30562;
   89
   90     (* set up event flag to wake up TRNSDB *)
   91     efn := QABFLAG;
   92
   93     (*set boolean field to queue type *)
   94     buf.typ := false;
   95
   96     (* put symbol into the buf.req field of the send buffer *)
   97     buf.ch := messageXq;
   98
   99     (* send the data to the TRNSDB task *)
  100     send(tsk, buf, efn, ids);
  101     if ids < 0 then
  102     begin
  103        writeln(tty, 'error in calling SEND in WQAB,  error code - ', ids);
  104        halt
  105     end;
  106
  107  {writeln(tty,' leaving wqab procedure')}
  108  end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000470
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000564
COMPILATION TIME 9 SECONDS

PAS WQAB,WQAB/P42-WQAB
```

```
 1  (*$m-*)
 2  (*****************************************************************************
 3
 4     NAME:        WTERMA
 5
 6     FUNCTION:    To write a TRNSDA error message on the terminal in case
 7                  illegal requests are made by TRNSDB or terminal A or to
 8                  write a query status message or one of the following
 9                  messages (successful): tconn, tcnnd, tdsnd, tnodi, texdi.
10
11     USAGE:       procedure wterma(messno : integer;
12                                   stateXa : state;
13                                   rreq : request);external;
14
15     INPUT PARAMETERS:    messno  - the type of message.
16                          stateXa - the state of TRNSDA.
17                          rreq    - the inputed string.
18
19     OUTPUT PARAMETERS:   NONE
20
21     MODULES CALLED:      (The symbol ' ' designates a primitive)
22
23                          'halt -
24                          'writeln
25
26     EXIT CONDITIONS:
27                          Normal exit conditions.
28
29     REMARKS:     NONE
30
31  *****************************************************************************)
32
33  const
34          REQUESTLEN = 6;
35
36  type
37          request = packed array[1..REQUESTLEN] of char;
38
39          state = (one, two, three, four);
40
41  procedure wterma(messno : integer ;stateXa : state ; rreq : request);
42
43  const                          { LOCAL VT100 TERMINAL CONSTANTS }
```

```
   44          revideo = '[7m';
   45          video   = '[0m';
   46          up      = '[A';
   47
   48    begin
   49
   50       (* put in reverse video display mode *)
   51       writeln(chr(27), revideo, chr(27), up);
   52
   53       if rreq = 'ustat ' then
   54       begin
   55         write(tty, 'NTRNSDA is in ');
   56           case stateXs of
   57
   58             one : write(tty, 'IDLE ');
   59             two : write(tty, 'OUT CONNECT PENDING ');
   60             three: write(tty, 'IN CONNECT PENDING ');
   61             four : write(tty, 'DATA TRANSFER READY ');
   62           end;
   63           writeln(tty, 'state.')
   64       end
   65       else case messno of
   66
   67          1 :  writeln (TTY,'tccon');
   68          2 :  writeln (TTY,'tcind');
   69          3 :  writeln (TTY,'tdind');
   70          4 :  writeln (TTY,'tnudi');
   71          5 :  writeln (TTY,'texdi');
   72          6 :  begin
   73            write(tty, 'NTRNSDA cannot accept a ', rreq, ' in ');
   74            case stateXs of
   75
   76              one : write(tty, 'IDLE ');
   77              two : write(tty, 'OUT CONNECT PENDING ');
   78              three: write(tty, 'IN CONNECT PENDING ');
   79              four : write(tty, 'DATA TRANSFER READY ');
   80            end;
   81            writeln(tty, 'state.')
   82        end
   83
   84       end;      (* case *)
   85
   86       (* put back in normal mode *)
```

```
   87       writeln(chr(27), video, chr(27), up);
   88
   89   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      002336
OUTERHOST DATA SIZE     000002
RESERVED STACK & HEAP   000520
COMPILATION TIME 17 SECONDS

PAS WTERMA,WTERMA/P42=WTERMA
```

APPENDIX C

DESCRIPTION OF ROUTINES
OF THE PROTOTYPE FOR
CLASS-0 TP SPECIFICATIONS
(User A's side only)

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4       NAME:         BUFFCLEAR
    5
    6       FUNCTION:     To clear the event flag corresponding to the passed integer value.
    7
    8       USAGE:        procedure buffclear(efn : efnn);
    9                                 .
   10                                 .
   11                                 .
   12                     buffclear(efn);
   13
   14       INPUT PARAMETERS:   efn              - the event flag number.
   15
   16       OUTPUT PARAMETERS:  NONE
   17
   18       MODULES CALLED:
   19                           >halt
   20                           >writeln
   21                           clref
   22
   23       EXIT CONDITIONS:
   24
   25                           If the retcode from the CLEF$ Directive is less
   26                           than zero then an error message is printed and
   27                           the event flag may not have been cleared and
   28                           the processor is halted using the 'halt' primitive.
   29
   30       REMARKS:
   31                           See RSX-11M V3.2 Executive Directives Manual on page 6-26 for
   32                           more information.
   33
   34   *****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:          BUFFGO
    5
    6      FUNCTION:      To set the event flag corresponding to the passed integer value.
    7
    8      USAGE:         procedure buffgo(efn : efnn);
    9
   10                                .
   11                                .
   12                     buffgo(efn);
   13
   14      INPUT PARAMETERS:    efn              - the event flag number.
   15
   16      OUTPUT PARAMETERS:   NONE
   17
   18      MODULES CALLED:
   19                           >halt
   20                           >writeln
   21                           setef
   22
   23      EXIT CONDITIONS:     Normal exit conditions.
   24                           if the retcode from the SETEF$ Directive is less
   25                           than zero then an error message is printed and the
   26                           event flag may not be set and the processor is halted
   27                           using the 'halt' primitive.
   28
   29      REMARKS:
   30                     See RSX-11M V3.2 Executive Directives Manual on page 6-123 for
   31                     more information.
   32
   33                     Also see chapter 3 of the Manual for more info. on event flags and
   34                     their use.
   35
   36  *****************************************************************************)
```

```
(*$m-*)
(*****************************************************************************
  NAME:          BUFFWAIT

  FUNCTION:      To wait until the event flag corresponding to the passed integer value
                 is set.

  USAGE:         procedure buffwait(efn : efnn);
                        .
                        .
                        .
                 buffwait(efn);

  INPUT PARAMETERS:    efn          - the event flag number.

  OUTPUT PARAMETERS:   NONE

  MODULES CALLED:
                       >halt
                       >writeln
                       waitfr

  EXIT CONDITIONS:     Normal exit conditions.

                       if the retcode from the WTSE$ Directive is less
                       than zero then an error message is printed and
                       the processor is halted using the 'halt' primitive.

  REMARKS:
                 See RSX-11M V3.2 Executive Directives Manual on page 6-168 for more
                 information.

                 One should be aware of the different types of event flags that
                 can be used for different purposes since for example if global
                 event flags are used for controlling events in a particular task
                 then one must make sure that these flags are not used somewhere
                 else in the system (see chapter 3 of Manual).

*****************************************************************************)
```

```
    1   (*$n-*)
    2   (*****************************************************************************
    3
    4      NAME:         FILLB
    5
    6      FUNCTION:     This procedure converts an integer to character format into the
    7                    given character field.
    8
    9      USAGE:        procedure fillb( b : integer;
   10                                     var c : char); extern;
   11
   12
   13
   14                   fillb(b, c);
   15
   16      INPUT PARAMETERS:    b        - the integer to convert.
   17
   18      OUTPUT PARAMETERS:   c        - the character field to be assigned.
   19
   20      MODULES CALLED:      (The symbol '>' designates a primitive)
   21
   22                           >chr
   23                           >halt
   24                           >writeln
   25
   26      EXIT CONDITIONS:
   27                           If the passed integer is > 256 or < 0 then
   28                           an error message is printed on the terminal
   29                           and the processor is halted.
   30
   31      REMARKS:             For Modularity and Portability reasons the primitive
   32                           'chr' should have been replaced by fillb in all the
   33                           modules in this system.  This should definitely be
   34                           considered for futur enhancements.
   35
   36  *****************************************************************************)
```

```
    1   (*$u-*)
    2   (*****************************************************************************
    3
    4     NAME:         FILLW
    5
    6     FUNCTION:     This procedure converts the passed integer to its equivalent
    7                   two characters LO and HI.
    8
    9     USAGE:        procedure fillw( b : integer;
   10                                    var c1 : char;
   11                                    var c2 : char); extern;
   12                                .
   13                                .
   14                                .
   15                  fillw(b, c1, c2);
   16
   17     INPUT PARAMETERS:    b       - the integer to convert.
   18
   19     OUTPUT PARAMETERS:   c1      - the LO field of the integer.
   20                          c2      - the HI field of the integer.
   21
   22     MODULES CALLED:      (The symbol '>' designates a primitive)
   23
   24                          >halt
   25                          >writeln
   26
   27     EXIT CONDITIONS:
   28                          Normal exit conditions
   29
   30     REMARKS:
   31                          It is to note here that the PDP-11 stores an integer's
   32                          hi and low bytes in inverse order in memory.  That is
   33                          LO-HI.
   34
   35  *****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         INITIAL
    5
    6      FUNCTION:     This module is used to create a dynamic region for
    7                    task TPRENTA by issuing a CRRG$ memory management directive.
    8
    9      USAGE:        procedure initial(var id : integer); extern;
   10                                .
   11                                .
   12                                .
   13                    initial(id);
   14
   15      INPUT PARAMETERS:    NONE
   16
   17      OUTPUT PARAMETERS:   id      - the region id.
   18
   19      MODULES CALLED:      (The symbol '>' designates a primitive)
   20
   21                           >halt
   22                           >writeln
   23                           crrg
   24
   25      EXIT CONDITIONS:
   26                           If there is an error detected in calling CRRG$, then
   27                           an error message identifying the error and the error
   28                           code is printed onto standard I/O and the processor
   29                           is then halted by using the 'halt' primitive.
   30
   31                           The return code of the CRRG$ routine is set to greater
   32                           or equal to 0 (zero) in case if successful completion
   33                           and to less than 0 in case of unsuccessful completion.
   34
   35                           For more information on the use of CRRG$ and of its
   36                           return codes, see pages 6-36, 6-38 of the RSX-11M V3.2
   37                           Executive Reference Manual.
   38
   39      REMARKS:             At the moment, there is no protection on the region.
   40                           One can easily remedy this by giving an appropriate
   41                           value to the region protection word.
   42
   43  *****************************************************************************)
```

```
(*$m-*)
(*****************************************************************************

   NAME:        INITBUF

   FUNCTION:    To initialize a TPDU with zeros in all of its fields.

   USAGE:       procedure  initbuf(var swndno : std);extern;
                           .
                           .
                           .
                initbuf(swndno);

   INPUT PARAMETERS:  NONE

   OUTPUT PARAMETERS:  swndno  - the record to be filled with zeros in all
                                 of its fields (initialization).


   MODULES CALLED:     ( '>' indicates a primitive )
                       >chr
                       >writeln

   EXIT CONDITIONS:    Normal condition.

   REMARKS:
                       The FILLB routine should probably have been used here
                       instead of chr.

*****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         INSPTPDU
    5
    6      FUNCTION:     To inspect a TPDU for erroneous parameter
    7                    codes or parameter values and returns the length of
    8                    the erroneous packet if an error is detected and sets
    9                    valid to FALSE, otherwise sets valid to TRUE.
   10
   11      USAGE:        procedure insptpdu(ptrwnd : pstd;
   12                                       var valid : boolean;
   13                                       var rjcse : integer;
   14                                       var err : errr); extern;
   15                                .
   16                                .
   17                                .
   18                    insptpdu(ptrwnd, valid, rjcse, err);
   19
   20      INPUT PARAMETERS:    ptrwnd  - the pointer to the TPDU to inspect.
   21
   22      OUTPUT PARAMETERS:   valid   - set to FALSE if error is detected and
   23                                     to TRUE otherwise.
   24                           rjcse   - the detected cause for rejection of TPDU.
   25                           err     - the length of the erroneous string
   26                                     which is sent back.
   27
   28      MODULES CALLED:      (The symbol '>' designates a primitive)
   29
   30                           >ord
   31                           >writeln
   32                           val
   33
   34
   35      EXIT CONDITIONS:
   36                           If an error is detected in the received buffer,
   37                           the length of the erroneous string
   38                           is returned with valid = FALSE
   39                           otherwise valid is set to TRUE.
   40
   41      REMARKS:
   42                           Length returned by this routine is up to and
   43                           including the first erroneous parameter or STRLEN
```

```
   44                        in case of successful inspection.
   45
   46                        At present, the inspection performed is not very
   47                        sophisticated since it is basically done by comparing
   48                        the different fields of a particullar TPDU with some
   49                        fixed global constants which represent the only value
   50                        acceptable.  For our purposes this method has proven to
   51                        be sufficient.
   52
   53   ************************************************************************)
```

```
(*$m-*)
{*****************************************************************************

    NAME:           NTAINIT

    FUNCTION:       This module is used to create a dynamic region for
                    task NTRNSDA by issuing a CRRG$ memory management directive.

    USAGE:          procedure ntainit(var id : integer); extern;
                                  .
                                  .
                                  .
                    ntainit(id);

    INPUT PARAMETERS:    NONE

    OUTPUT PARAMETERS:   id      - the region id.

    MODULES CALLED:      (The symbol '>' designates a primitive)

                         >halt
                         >writeln
                         crrg

    EXIT CONDITIONS:
                         If there is an error detected in calling CRRG$, then
                         an error message identifying the error and the error
                         code is printed onto standard I/O and the processor
                         is then halted by using the 'halt' primitive.

                         The return code of the CRRG$ routine is set to greater
                         or equal to 0 (zero) in case if successful completion
                         and to less than 0 in case of unsuccessful completion.

                         For more information on the use of CRRG$ and of its
                         return codes, see pages 6-36, 6-38 of the RSX-11M V3.2
                         Executive Reference Manual.

    REMARKS:             At the moment, there is no protection on the region.
                         One can easily remedy this by giving an appropriate
                         value to the region protection word.

*****************************************************************************}
```

```
1   (*$m-*)
2   (*****************************************************************************
3
4      NAME:        NTAREC
5
6      FUNCTION:    To wait until some information is sent to the NTRNSDA task
7                   via send data or send by reference, and receives it via
8                   receive data or receive by reference.
9
10     USAGE:       procedure ntarec(var ptrwnd : pstd;
11                                   var nXreu : netXreq;
12                                   var queue : boolean;
13                                   var messXtprenta : boolean); extern;
14                              .
15                              .
16                              .
17                  ntarec(ptrwnd, nXreq, queue, messXtprenta);
18
19
20     INPUT PARAMETERS     NONE
21
22     OUTPUT PARAMETERS:   ptrwnd  - the pointer to receive by reference buffer.
23                          nXreu   - receive data string or receive by reference
24                                    string.
25                          queue   - true if a receive data was performed
26                                    false if a receive by reference was done.
27                          messXtprenta- true if data was sent by TPRENTA task
28                                        false if data was sent by NTRNSDB task.
29
30     MODULES CALLED:      (The symbol '>' desisnates a primitive)
31
32                          >chr
33                          >halt
34                          >writeln
35                          buffwait
36                          craw
37                          dtrg
38                          mark
39                          receiv
40                          rref
41
42     EXIT CONDITIONS:
43                          If there are any errors detected in calling CRAW$, DTRG$,
```

```
    44                          MARK$, RECEIV$ or RREF$, then an error message identifying
    45                          the error and error code is printed onto standard I/O and
    46                          the processor is then halted by usint the 'halt' primitive.
    47
    48                          The return code of all these system directives, is set to
    49                          greater or equal to 0(zero) in case of successful completion
    50                          and to less than 0 in case of unsuccessful completion.
    51
    52                          For more information on the use of these system routines
    53                          and their return codes, see the RSX-11M V3.2 Executive
    54                          Reference Manual.
    55
    56      REMARKS:        When there is a receive by reference, the newly attached region
    57                      is detached in order to prevent depletion of the memory pool.
    58                      See page 6-108 of Executive Reference Manual for more details.
    59
    60  ********************************************************************************)
```

```
(*****************************************************************************

   NAME:          NTRNSDA (MAIN)

   FUNCTION:      Network transducer for side 'A'.

   USAGE:         from MCR : 'RUN NTRNSDA /TASK=TRNSDA /INC=100.'

   INPUT PARAMETERS:    NONE

   OUTPUT PARAMETERS:   NONE

   MODULES CALLED:      (The symbol '>' designates a primitive)

                        >chr
                        >new
                        >ord
                        >write
                        >writeln
                        buffclear
                        buffgo
                        buffwait
                        ntainit
                        ntarec
                        sdtprenta
                        wuab
                        wmess

   EXIT CONDITIONS:
                        This program has been designed to run indefinitely until
                        the task into which it runs is specifically aborted with
                        the MCR command 'ABORT TRNSDA'.

                        If any errors are detected by one of the routines called,
                        then the routine itself will print an error message and
                        halt the processor by using the 'halt' primitive.

   REMARKS:
                        See Ref. (22) for more details.

*****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         PREPTPDU
    5
    6      FUNCTION:     To prepare a TPDU accordins to the TPDU code.
    7
    8      USAGE:        procedure preptpdu (ptrwnd : pstd;
    9                                        var dstref : arri;
   10                                        var srcref : arri;
   11                                        rjcse  : integer;
   12                                        err : errr;
   13                                        cod : integer);
   14                                .
   15                                .
   16                                .
   17                   preptpdu(ptrwnd, dstref, srcref, rjcse, err, cod);
   18
   19      INPUT PARAMETERS:     ptrwnd  - the pointer to the packet to be filled.
   20                            dstref  - destination address.
   21                            srcref  - source address.
   22                            rjcse   - cause for rejection of TPDU.
   23                            err     - length of the erroneous TPDU string.
   24                            cod     - the type of TPDU packet.
   25
   26      OUTPUT PARAMETERS:    ptrwnd  - the pointer to the TPDU.
   27                            dstref  - destination address.
   28                            srcref  - source address.
   29
   30      MODULES CALLED:       (The symbol '>' designates a primitive)
   31
   32                            >chr
   33                            >writeln
   34
   35      EXIT CONDITIONS:
   36                            Normal exit conditions.
   37
   38      REMARKS:
   39                   Presently, most of the filling is done in TERMA.  For futur
   40                   enhancements, the filling should be done by this module.
   41
   42                   See the TPRENTA module for more information about the format
   43                   of the different TPDUs.
```

```
  44
  45  ********************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4     NAME:       .READuDATA
    5
    6     FUNCTION:   To read in a string from standard input into the appropriate
    7                 field of a particular packet, record its length and put it into
    8                 the correct length field of this packet.
    9
   10     USAGE:      procedure  readUdata(var w : std);extern;
   11                             .
   12                             .
   13                             .
   14                 readUdata(w);
   15
   16     INPUT PARAMETERS:   NONE
   17
   18     OUTPUT PARAMETERS:  w       - the packet to be filled with a string and its
   19                                   associated length.
   20
   21     MODULES CALLED:
   22                         >break
   23                         >eoln
   24                         >read
   25                         >readln
   26                         >writeln
   27                         fillb
   28
   29     EXIT CONDITIONS:    Normal condition.
   30
   31     REMARKS:
   32                 The inputed line is truncated to STRLEN, so care should
   33                 be taken not to exceed this value.
   34
   35                 The user (TERMINAL A) should make sure that his/her terminal
   36                 buffer is at least >= STRLEN.  This can be checked by issuing
   37                 the MCR command 'SET /BUF=TI:'.  If the buffer is too small,
   38                 then this can be altered by the MCR command 'SET /BUF=TI:200.'
   39                 if a buffer of 200 bytes is desired for example.
   40
   41                 Unpredictable behavior results when non-printable characters
   42                 , such as escape or end-of-file characters, are inputed by
   43                 this module.
```

```
   44
   45  ********************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:        READTERM
    5
    6      FUNCTION:    To read in a string from standard input and record its length.
    7
    8      USAGE:       procedure request(var buffer : request ; var len : integer);extern;
    9
   10
   11
   12                  readterm(buffer, len);
   13
   14      INPUT PARAMETERS:    NONE
   15
   16      OUTPUT PARAMETERS:   buffer          - the string buffer.
   17                           len             - the length of the string buffer.
   18
   19      MODULES CALLED:
   20                           >break
   21                           >eoln
   22                           >read
   23                           >readln
   24                           >writeln
   25
   26      EXIT CONDITIONS:     Normal condition.
   27
   28      REMARKS:
   29                  The inputed line is truncated to REQLEN, so care should
   30                  be taken not to exceed this value.
   31
   32                  It is not recommended to input non-printable characters using
   33                  this module since unpredictable behavior can result when these
   34                  strings are outputed.
   35
   36  *****************************************************************************)
```

```
1   (*$m-*)
2   (*****************************************************************************
3
4       NAME:         RECCA.
5
6       FUNCTION:     To wait until some information is sent to the TPRENTA task
7                     via send data or send by reference, and receives it via
8                     receive data or receive by reference.
9
10      USAGE:        procedure recca(var ptrwnd : pstd;
11                                    var nXreq : netXreq;
12                                    var queue : boolean;
13                                     var messXterma : boolean); extern;
14                                .
15                                .
16                                .
17                    recca(ptrwnd, nXreq, queue, messXterma);
18
19
20      INPUT PARAMETERS      None
21
22      OUTPUT PARAMETERS:    ptrwnd  - pointer to received by reference buffer.
23                            nXreq   - receive data or reference string.
24                            queue   - true if a receive data was performed
25                                      false if a receive by reference was done.
26                            messXterma- true if data was sent by TERMA task
27                                      false if data was sent by NTRNSDA task.
28
29      MODULES CALLED:       (The symbol '>' designates a primitive)
30
31                            >chr
32                            >halt
33                            >writeln
34                            buffwait
35                            craw
36                            dtrg
37                            mark
38                            receiv
39                            rref
40
41      EXIT CONDITIONS:
42                            If there are any errors detected in calling CRAW$, DTRG$
43                            MARK$, RECEIV$ or RREF$, then an error message identifying
```

```
44                          the error and error code is printed onto standard I/O and
45                          the processor is then halted by using the 'halt' primitive.
46
47                          The return code of all these system directives, is set to
48                          greater or equal to 0(zero) in case of successful completion
49                          and to less than 0 in case of unsuccessful completion.
50
51                          For more information on the use of these system routines
52                          and their return codes, see the RSX-11M V3.2 Executive
53                          Reference Manual.
54
55      REMARKS:        When there is a receive by reference, the newly attached region
56                      is detached in order to prevent depletion of the memory pool.
57                      See page 6-108 of Executive Reference Manual for more details.
58
59  ********************************************************************************)
```

```
 1  (*$m-*)
 2  (*****************************************************************************
 3
 4     NAME:          SDTPRENTA
 5
 6     FUNCTION:      To send information to task TPRENTA via system directives
 7                    send data (SDAT$) or send by reference (SREF$).
 8
 9     USAGE:         procedure sdtprenta(ptrwnd : ptid;
10                                        var wndno : wndnoo;
11                                        nXreq : netXreq;
12                                        id : integer;
13                                        bol : boolean); extern;
14
15                                .
16                                .
17                    sdtprenta(ptrwnd, wndno, nXreq, id, bol);
18
19     INPUT PARAMETERS:    ptrwnd  - pointer to the buffer to be sent.
20                          wndno   - the window number to be sent if bol = false.
21                          nXreq   - the string to be sent.
22                          id      - the region id.
23                          bol     - the boolean switch.
24
25     OUTPUT PARAMETERS:   wndno   - the next window number.
26
27     MODULES CALLED:      (The symbol '>' designates a primitive)
28
29                          >halt
30                          >writeln
31                          craw
32                          sref
33                          sdat
34
35     EXIT CONDITIONS:
36
37                          If there are any errors detected in calling CRAW$, SDAT$,
38                          or SREF$, then an error message identifying the error and
39                          error code is printed onto standard I/O and the processor
40                          is then halted by using the 'halt' primitive.
41
42                          The return code of all these system directives, is set or
43                          greater or equal to 0 (zero) in case of successful completion
```

```
                        and less than 0 in case of unsuccessful completion.

                        For more information on the use of these system routines
                        and their return codes, see the RSX-11M V3.2 Executive
                        Reference Manual.

    REMARKS:

                        See TSEND routine for information on method of circular
                        window queues and its restrictions.

*****************************************************************************)
```

```
(*$m-*)
(*****************************************************************************)

   NAME:        SENDD

   FUNCTION:    To send information to task NTRNSDA via system directives
                send data (SDAT$) or send by reference (SREF$).

   USAGE:       procedure sendd(ptrwnd : pstd;
                                var wndno : wndnoo;
                                nXreq : netXreq;
                                id : integer;
                                bol : boolean); extern;

                sendd(ptrwnd, wndno, nXreq, id, bol);

   INPUT PARAMETERS:    ptrwnd  - the pointer to the buffer to be sent.
                        wndno   - the window number to be used.
                        nXreq   - the string to be sent if bol = true.
                        id      - the region id.
                        bol     - the boolean switch.

   OUTPUT PARAMETERS:   wndno   - the next window number.

   MODULES CALLED:      (The symbol '>' designates a primitive)

                        >halt
                        >writeln
                        craw
                        sref
                        sdat

   EXIT CONDITIONS:
                        If there are any errors detected in calling CRAW$, SDAT$
                        or SREF$, then an error message identifying the error and
                        error code is printed onto standard I/O and the processor
                        is then halted by using the 'halt' primitive.

                        The return code of all these system directives, is set or
                        greater or equal to 0 (zero) in case of successful completion
                        and less than 0 in case of unsuccessful completion.
```

```
   44
   45                        For more information on the use of these system routines
   46                        and their return codes, see the RSX-11M V3.2 Executive
   47                        Reference Manual.
   48
   49      REMARKS:     NONE
   50
   51    ********************************************************************************)
```

```
    1   (*****************************************************************************
    2
    3     NAME:         TERMA (MAIN)
    4
    5     FUNCTION:     This module is used to continuously READ information from
    6                   USER A (TERMINAL A) and to send this information to the TPRENTA
    7                   task.  This information is obtained from terminal input, in the
    8                   form of user primitive strings and also data strings (see tnodr
    9                   and tdreq primitives), and also from certain text files which
   10                   contain information related to the selected user primitive.
   11
   12     USAGE:        from MCR : 'RUN TERMA /TASK=TERMA /INC-100.'
   13
   14     INPUT PARAMETERS:    NONE
   15
   16     OUTPUT PARAMETERS:   NONE
   17
   18     MODULES CALLED:      (The symbol '>' designates a primitive)
   19
   20                          >chr
   21                          >halt
   22                          >ioresult
   23                          >read
   24                          >readln
   25                          >reset
   26                          >writeln
   27                          buffclear
   28                          buffgo
   29                          buffwait
   30                          fillb
   31                          fillw
   32                          initbuf
   33                          readterm
   34                          readUdata
   35                          tinit
   36                          tsend
   37
   38      EXIT CONDITIONS:
   39                          This program has been designed to run indefinitely until
   40                          the task into which it runs is specifically aborted with
   41                          the MCR command 'ABORT TERMA'.
   42
   43                          Whenever there is an error detected, an error message
```

```
   44                        identifying the error and error code is printed onto
   45                        standard I/O and the processor is then halted by using
   46                        the 'halt' primitive.  Here is a list of the possible
   47                        errors :
   48
   49                        - file 'TCREQ.TXT' could not be reset properly,
   50                        - file 'TCRES.TXT' could not be reset properly
   51                        - file 'TDREQ.TXT' could not be reset properly.
   52
   53                        If any errors are detected by one of the routines called,
   54                        then the routine itself will print an error message and
   55                        halt the processor.
   56
   57      REMARKS:
   58
   59                        The files 'TCREQ.TXT', 'TCRES.TXT' and 'TDREQ.TXT' are
   60                        formatted in accordance with the specifications of this
   61                        module. Any changes in the format of this module or
   62                        of those text files should be done very carefully.
   63    ***************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:        TINIT
    5
    6      FUNCTION:    This module is used to create a dynamic region
    7                   for task TERMA by issuing a CRRG$ memory management directive.
    8
    9
   10      USAGE:       procedure tinit(var id : integer); extern;
   11                                .
   12                                .
   13                                .
   14                   tinit(id);
   15
   16      INPUT PARAMETERS:    NONE
   17
   18      OUTPUT PARAMETERS:   id      - the region id.
   19
   20      MODULES CALLED:      (The symbol '>' designates a primitive)
   21
   22                           >halt
   23                           >writeln
   24                           crrg
   25
   26      EXIT CONDITIONS:
   27                           If there is an error detected in calling CRRG$, then
   28                           an error message identifying the error and error code
   29                           is printed onto standard I/O and the processor is then
   30                           halted by using the 'halt' primitive.
   31
   32                           The return code of the CRRG$ routine is set to greater
   33                           or equal to 0 (zero) in case of successful completion and to
   34                           less than 0 in case of unsuccessful completion.
   35
   36                           For more information on the use of CRRG$ and of its
   37                           return codes, see pages 6-36,.6-38 of the RSX-11M V3.2
   38                           Executive Reference Manual.
   39
   40      REMARKS:             At the moment, there is no protection on the region.
   41                           One can easily remedy this by giving an appropriate
   42                           value to the region protection word.
   43
```

```
   44  ********************************************************************************)
```

```
    1   (*****************************************************************************
    2
    3       NAME:         TPRENTA (MAIN)
    4
    5       FUNCTION:     To serve as Transport Protocol Entity for USER 'A'.
    6
    7       USAGE:        from MCR : 'RUN TPRENTA /TASK=TPRENA /INC=100.'
    8
    9       INPUT PARAMETERS:     NONE
   10
   11       OUTPUT PARAMETERS:    NONE
   12
   13       MODULES CALLED:       (The symbol '>' designates a primitive)
   14
   15                             >chr
   16                             >new
   17                             >ord
   18                             >write
   19                             >writeln
   20                             buffclear
   21                             buffgo
   22                             buffwait
   23                             initial
   24                             insptpdu
   25                             preptpdu
   26                             recca
   27                             sendd
   28                             werra
   29                             wterma
   30
   31       EXIT CONDITIONS:
   32                             This program has been designed to run indefinitely until
   33                             the task into which it runs is specifically aborted with
   34                             the MCR command 'ABORT TPRENA'.
   35
   36                             If an error is detected by one of the routines called,
   37                             then the routine itself will print an error message and
   38                             halt the processor by using the 'halt' primitive.
   39
   40       REMARKS:
   41                             The format of the TPDUs is different from the standard
   42                             format since the PDP-11 swaps the Hi and LO bytes of a
   43                             word in memory.  So internally speaking (in memory) the
```

```
   44                          TPDUs are according to standards.
   45
   46   ********************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:       TSEND
    5
    6      FUNCTION:   To send a buffer of information to task TPRENTA, by using the
    7                  send by reference directive SREF$.
    8
    9      USAGE:      procedure tsend(swndno : std;
   10                                  var wndno : wndnoo;
   11                                  id : integer); extern;
   12                             .
   13                             .
   14                             .
   15                 sendd(swndno, wndno, id);
   16
   17      INPUT PARAMETERS:   swndno  - the buffer to send by reference.
   18                          wndno   - the window number to be used(1 to 7).
   19                          id      - the region id.
   20
   21      OUTPUT PARAMETERS:  wndno   - the new window number.
   22
   23      MODULES CALLED:     (The symbol '>' designates a primitive)
   24
   25                          >halt
   26                          >writeln
   27                          craw
   28                          sref
   29
   30      EXIT CONDITIONS:
   31                          If there are any errors detected in calling CRAW$ or
   32                          SREF$, then an error message identifying the error
   33                          and error code is printed onto standard I/O and the
   34                          processor is then halted by using the 'halt' primitive.
   35
   36                          The return code of both these system directives, is set
   37                          to greater or equal to 0(zero) in case of successful
   38                          completion and to less than 0 in case of unsuccessful
   39                          completion.
   40
   41                          For more information on the use of CRAW$ and SREF$ and
   42                          their return codes, see pages 6-31..6-34 and 6-138..6-140
   43                          of the RSX-11M V3.2 Executive Reference Manual.
```

```
44
45      REMARKS:
46                          The method used in this module involves putting information
47                          in the region windows and sending these by reference.
48                          Each call to this routine involves sending the next
49                          window in line (eg. the next window in the region or
50                          window number one if the last window(ie. no 7) was used
51                          on the previous call).  This strategy guarantees that at
52                          least 7 'SENDDs' can be done before any of the data gets
53                          overwritten by new data.
54
55  ********************************************************************************)
```

```
    1   (*$n-*)
    2   (*****************************************************************************
    3
    4      NAME:         VAL
    5
    6      FUNCTION:     This function takes two characters c1(LO) and c2(HI) and
    7                    returns the integer that the combination of HI-LO forms.
    8
    9      USAGE:        i  : integer;
   10
   11                   function val(c1 : char; c2 : char) : integer ; extern;
   12
   13
   14
   15                   i := val(c1, c2);
   16
   17      INPUT PARAMETERS:    c1      - the LO character.
   18                           c2      - the HI character.
   19
   20      OUTPUT PARAMETERS:   (returned)- the integer formed by HI-LO.
   21
   22      MODULES CALLED:      (The symbol '>' designates a primitive)
   23
   24                           >halt
   25                           >writeln
   26
   27      EXIT CONDITIONS:
   28                           Normal exit conditions (top_down)
   29      REMARKS:
   30                           One should be aware that the hi and low bytes of a word
   31                           are positioned as LO-HI in memory on the PDP-11.
   32
   33  *****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4       NAME:         WERRA
    5
    6       FUNCTION:     Write an appropriate error message on User A's terminal
    7                     due to an error situation in TPRENTA.
    8
    9       USAGE:        procedure werra(ptrwnd : psid;
   10                                    rjcse : integer;
   11                                    mes : wmessXerr); extern;
   12                                .
   13                                .
   14                                .
   15                    werra(ptrwnd, rjcse, mes);
   16
   17       INPUT PARAMETERS:    ptrwnd  - pointer to received TPDU.
   18                            rjcse   - cause for rejection of TPDU.
   19                            mes     - the error message number.
   20
   21       OUTPUT PARAMETERS:   NONE
   22
   23       MODULES CALLED:      (The symbol '>' designates a primitive)
   24
   25                            >ord
   26                            >writeln
   27
   28       EXIT CONDITIONS:
   29                            Normal exit conditions.
   30
   31       REMARKS:
   32                            The TPRENTA module should be studied for a better
   33                            understanding of the error encountered.
   34
   35  *****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4       NAME:         WMESS
    5
    6       FUNCTION:     To write a NTRNSDA error message on the terminal in case
    7                     illegal requests are made by TPRENTA or NTRNSDB.
    8
    9       USAGE:        procedure wmess(messno : integer;
   10                                    stateXa : state;
   11                                    rreq : netXreq); extern;
   12
   13       INPUT PARAMETERS:    messno  - the type of message.
   14                            stateXa - the state of NTRNSDA.
   15                            rreq    - the inputed string.
   16
   17       OUTPUT PARAMETERS:   NONE
   18
   19       MODULES CALLED:      (The symbol '>' designates a primitive)
   20                            >halt
   21                            >writeln
   22
   23       EXIT CONDITIONS:
   24                            Normal exit conditions.
   25
   26       REMARKS:    NONE
   27
   28  *****************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         WQAB
    5
    6      FUNCTION:     To send information to task NTRNSDB via system directives
    7                    send data (SDAT$) or send by reference (SREF$).
    8
    9      USAGE:        procedure wqab(ptrwnd : pstd;
   10                                   var wndno : wndnoo;
   11                                   nXreq : netXreq;
   12                                   id : integer;
   13                                   bol : boolean); extern;
   14                          .
   15                          .
   16                          .
   17                   wqab(ptrwnd, wndno, nXreq, id, bol);
   18
   19      INPUT PARAMETERS:    ptrwnd  - the pointer to the buffer to be sent.
   20                           wndno   - the window number to be used if bol = false.
   21                           nXreq   - the string to be sent.
   22                           id      - the region id.
   23                           bol     - the boolean switch.
   24
   25      OUTPUT PARAMETERS:   wndno   - the next window number.
   26
   27      MODULES CALLED:      (The symbol '>' designates a primitive)
   28
   29                           >halt
   30                           >writeln
   31                           qraw
   32                           sref
   33                           sdat
   34
   35      EXIT CONDITIONS:
   36                           If there are any errors detected in calling QRAW$, SDAT$
   37                           or SREF$, then an error message identifying the error and
   38                           error code is printed onto standard I/O and the processor
   39                           is then halted by using the 'halt' primitive.
   40
   41                           The return code of all these system directives, is set or
   42                           greater or equal to 0 (zero) in case of successful completion
   43                           and less than 0 in case of unsuccessful completion.
```

```
   44
   45                        For more information on the use of these system routines
   46                        and their return codes, see the RSX-11M V3.2 Executive
   47                        Reference Manual.
   48
   49      REMARKS:     NONE
   50
   51    ********************************************************************************)
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         WTERMA
    5
    6      FUNCTION:     Write an appropriate legal message on the terminal A
    7                    which could be TCIND, TDIND, TNODI or TCCON with
    8                    appropriate parameter values for the above primitives,
    9                    as per ISO Transport Protocol Specs. Apr. 1983.
   10
   11      USAGE:        procedure wterma(ptrwnd : pstd;
   12                                     mes : wmessXsts); extern;
   13                                .
   14                                .
   15                                .
   16                    wterma(ptrwnd, mes);
   17
   18      INPUT PARAMETERS:    ptrwnd  - the pointer to the TPDU received.
   19                           mes     - the message number.
   20
   21      OUTPUT PARAMETERS:   NONE
   22
   23      MODULES CALLED:      (The symbol '>' desisnates a primitive)
   24
   25                           >char
   26                           >ord
   27                           >writeln
   28                           val
   29
   30      EXIT CONDITIONS:
   31                           Normal exit conditions.
   32
   33      REMARKS:
   34                    A lot of the code is for generating a nice display on a
   35                    VT100 compatible terminal.
   36
   37                    Most of the fields in a TPDU are outputed using the 'ord'
   38                    primitive.  This is because the fields are stored in byte
   39                    (character) format.
   40
   41                    The VAL function is used to associate two bytes together and
   42                    form an integer.
   43
```

```
   44   ********************************************************************************)
```

APPENDIX D

PASCAL CODE OF ROUTINES
OF THE PROTOTYPE FOR
CLASS-0 TP SPECIFICATIONS
(User A's side only)

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:        BUFFCLEAR
    5
    6      FUNCTION:    To clear the event flag corresponding to the passed integer value.
    7
    8      USAGE:       procedure buffclear(efn : efnn);
    9                                 .
   10                                 .
   11                                 .
   12                   buffclear(efn);
   13
   14      INPUT PARAMETERS:    efn              - the event flag number.
   15
   16      OUTPUT PARAMETERS:   NONE
   17
   18      MODULES CALLED:
   19                           >halt
   20                           >writeln
   21                           clref
   22
   23      EXIT CONDITIONS:
   24
   25                           If the retcode from the CLEF$ Directive is less
   26                           than zero then an error message is printed and
   27                           the event flag may not have been cleared and
   28                           the processor is halted using the 'halt' primitive.
   29
   30      REMARKS:
   31                           See RSX-11M V3.2 Executive Directives Manual on page 6-26 for
   32                           more information.
   33
   34  *****************************************************************************)
   35
   36  type
   37          efnn = integer;
   38
   39  procedure buffclear(efn : efnn);
   40
   41  var  ids :integer;
   42
   43  procedure clref(var efn : integer; var ids : integer); extern(fortran);
```

```
PASCAL  PDP-11  VERSION  6.07        1908-08-04      03:23:55          PAGE   2
BUFFCLEAR.PAS                          BUFFCLEAR

   44
   45   begin
   46   { writeln(tty, 'entering buffclear routine');}
   47
   48     clref(efn, ids);       (* call CLEF$ *)
   49     if ids < 0 then                (* check if retcode is less than 0 *)
   50     begin
   51       writeln(' error in using clref. Error code =', ids);
   52       halt
   53     end
   54
   55   { writeln(tty, 'leaving buffclear routine')}
   56   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      000256
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 6 SECONDS

PAS BUFFCLEAR,BUFFCLEAR/P42=BUFFCLEAR
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:        BUFFGO
    5
    6      FUNCTION:    To set the event flag corresponding to the passed integer value.
    7
    8      USAGE:       procedure buffgo(efn : efnn);
    9                                 .
   10                                 .
   11                                 .
   12                   buffgo(efn);
   13
   14      INPUT PARAMETERS:    efn          - the event flag number.
   15
   16      OUTPUT PARAMETERS:   NONE
   17
   18      MODULES CALLED:
   19                           >halt
   20                           >writeln
   21                           setef
   22
   23      EXIT CONDITIONS:     Normal exit conditions.
   24                           if the retcode from the SETEF$ Directive is less
   25                           than zero then an error message is printed and the
   26                           event flag may not be set and the processor is halted
   27                           using the 'halt' primitive.
   28
   29      REMARKS:
   30                   See RSX-11M V3.2 Executive Directives Manual on page 6-123 for
   31                   more information.
   32
   33                   Also see chapter 3 of the Manual for more info. on event flags and
   34                   their use.
   35
   36  *****************************************************************************)
   37
   38  type
   39          efnn = integer;
   40
   41  procedure buffgo(efn : efnn);
   42
   43  var  ids :integer;
```

```
   44
   45   procedure setef(var efn:integer; var ids : integer); extern(fortran);
   46
   47   begin
   48   { writeln(tty, 'entering buffgo routine');}
   49
   50     setef(efn, ids);      (* call the SETF$ Directive *)
   51     if ids < 0 then              (* check if retcode is less than 0 *)
   52     begin
   53       writeln(' error in setef, Error code = ', ids);
   54       halt
   55     end
   56
   57   { writeln(tty, 'leaving buffgo routine');}
   58   end.

   NO ERROR DETECTED
   TOTAL PROGRAM SIZE      000250
   OUTERMOST DATA SIZE     000002
   RESERVED STACK & HEAP   000522
   COMPILATION TIME 5 SECONDS

   PAS BUFFGO,BUFFGO/P42=BUFFGO
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         BUFFWAIT
    5
    6      FUNCTION:     To wait until the event flag corresponding to the passed integer value
    7                    is set.
    8
    9      USAGE:        procedure buffwait(efn : efnn);
   10                                 .
   11                                 .
   12                                 .
   13                    buffwait(efn);
   14
   15      INPUT PARAMETERS:    efn              - the event flag number.
   16
   17      OUTPUT PARAMETERS:   NONE
   18
   19      MODULES CALLED:
   20                           >halt
   21                           >writeln
   22                           waitfr
   23
   24      EXIT CONDITIONS:     Normal exit conditions.
   25
   26                           if the retcode from the WTSE$ Directive is less
   27                           than zero then an error message is printed and
   28                           the processor is halted using the 'halt' primitive.
   29
   30      REMARKS:
   31                    See RSX-11M V3.2 Executive Directives Manual on page 6-168 for more
   32                    information.
   33
   34                    One should be aware of the different types of event flags that
   35                    can be used for different purposes since for example if global
   36                    event flags are used for controlling events in a particular task
   37                    then one must make sure that these flags are not used somewhere
   38                    else in the system (see chapter 3 of Manual).
   39
   40  *****************************************************************************)
   41
   42  type
   43          efnn = integer;
```

```
   44
   45   procedure buffwait(efn : efnn);
   46
   47   var ids : integer;
   48
   49   procedure waitfr(var efn:integer; var ids : integer); extern(fortran);
   50
   51   begin
   52   { writeln(tty, 'entering buffwait routine ');}
   53
   54     waitfr(efn, ids);     (* wait for flag to be set *)
   55
   56     if ids < 0 then              (* was it successful ? *)
   57     begin (* no *),
   58       writeln(' error in waitfr. Error code = ', ids);
   59       halt
   60     end
   61
   62   { writeln(tty, 'leaving buffwait routine ');}
   63   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000252
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 7 SECONDS

PAS BUFFWAIT,BUFFWAIT/P42=BUFFWAIT
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         FILLB
    5
    6      FUNCTION:     This procedure converts an integer to character format into the
    7                    given character field.
    8
    9      USAGE:        procedure fillb( b : integer;
   10                                     var c : char); extern;
   11                              .
   12                              .
   13                              .
   14                    fillb(b, c);
   15
   16      INPUT PARAMETERS:    b        - the integer to convert.
   17
   18      OUTPUT PARAMETERS:   c        - the character field to be assigned.
   19
   20      MODULES CALLED:      (The symbol ">" designates a primitive)
   21
   22                           >chr
   23                           >halt
   24                           >writeln
   25
   26      EXIT CONDITIONS:
   27                           If the passed integer is > 256 or < 0 then
   28                           an error message is printed on the terminal
   29                           and the processor is halted.
   30
   31      REMARKS:             For Modularity and Portability reasons the primitive
   32                           'chr' should have been replaced by fillb in all the
   33                           modules in this system.  This should definitely be
   34                           considered for futur enhancements.
   35
   36  *****************************************************************************)
   37
   38  procedure fillb( b : integer;
   39                   var c : char);
   40
   41  begin
   42  { writeln(tty, 'entering fillb subroutine');  }
   43    if ((b > 256) or (b < 0)) then
```

```
   44   begin
   45     writeln(tty, 'illegal passed integer in fillb routine');
   46     halt
   47   end;
   48   c := chr(b)
   49
   50   { writeln(tty, 'leaving fillb subroutine'); }
   51 end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000264
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000520
COMPILATION TIME 6 SECONDS

PAS FILLB,FILLB/P42-FILLB
```

```
    1   (*$a-*)
    2   (*****************************************************************************
    3
    4      NAME:         FILLW
    5
    6      FUNCTION:     This procedure converts the passed integer to its equivalent
    7                    two characters LO and HI.
    8
    9      USAGE:        procedure fillw( b : integer;
   10                                     var c1 : char;
   11                                     var c2 : char); extern;
   12                                 .
   13                                 .
   14                                 .
   15                   fillw(b, c1, c2);
   16
   17      INPUT PARAMETERS:    b        - the integer to convert.
   18
   19      OUTPUT PARAMETERS:   c1       - the LO field of the integer.
   20                           c2       - the HI field of the integer.
   21
   22      MODULES CALLED:      (The symbol '>' designates a primitive)
   23
   24                           >halt
   25                           >writeln
   26
   27      EXIT CONDITIONS:
   28                           Normal exit conditions
   29
   30      REMARKS:
   31                           It is to note here that the PDP-11 stores an integer's
   32                           hi and low bytes in inverse order in memory.  That is
   33                           LO-HI.
   34
   35  *****************************************************************************)
   36
   37  procedure fillw( b : integer;
   38                   var c1 : char;
   39                    var c2 : char);
   40
   41  type
   42          trick = record
   43                  (* variant part *)
```

```
    44                  case bol : boolean of
    45                          false : ( w : integer);
    46                          true  : ( c : packed array [1..2] of char)
    47                  end;
    48  var
    49          tt      : trick;
    50
    51  begin
    52  { writeln(tty, 'entering fillw subroutine'); }
    53
    54    tt.bol := false;
    55    tt.w := b;    (* put integer in integer part of variant record *)
    56    tt.bol := true;
    57    c1 := tt.c[1]; (* seperate the integer into 2 character fields *)
    58    c2 := tt.c[2]
    59
    60  { writeln(tty, 'leaving fillw subroutine') }
    61  end.
```



```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000240
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000524
COMPILATION TIME 5 SECONDS

PAS FILLW,FILLW/P42=FILLW
```

```
(*$m-*)
(*****************************************************************************

    NAME:           INITIAL

    FUNCTION:       This module is used to create a dynamic region for
                    task TPRENTA by issuing a CRRG$ memory management directive.

    USAGE:          procedure initial(var id: integer); extern;

                    initial(id);

    INPUT PARAMETERS:    NONE

    OUTPUT PARAMETERS:   id      - the region id.

    MODULES CALLED:      (The symbol '>' designates a primitive)

                         >halt
                         >writeln
                         crrg

    EXIT CONDITIONS:
                         If there is an error detected in calling CRRG$, then
                         an error message identifying the error and the error
                         code is printed onto standard I/O and the processor
                         is then halted by using the 'halt' primitive.

                         The return code of the CRRG$ routine is set to greater
                         or equal to 0 (zero) in case if successful completion
                         and to less than 0 in case of unsuccessful completion.

                         For more information on the use of CRRG$ and of its
                         return codes, see pages 6-36, 6-38 of the RSX-11M V3.2
                         Executive Reference Manual.

    REMARKS:             At the moment, there is no protection on the region.
                         One can easily remedy this by giving an appropriate
                         value to the region protection word.

*****************************************************************************)
```

```
   44
   45   {$I+TPRENTA.DEF}
```

```
    1
    2   const                          { GLOBAL CONSTANTS }
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           STRLEN     = 118;
    6           NETREQLEN  = 6;
    7           CRCODE     = 224;
    8           CCCODE     = 208;
    9           DTCODE     = 240;
   10           DRCODE     = 128;
   11           NDRCODE    = 120;
   12           ERCODE     = 112;
   13           APRNO      = 7;
   14           WNDSZ      = 3;
   15           REGSZ      = 21;  { NOPTR times WNDSZ }
   16           NOPTR      = 7;
   17           DUMLEN     = 10;
   18           DUMMLEN    = 5;
   19           NODATFLG   = 20;
   20           IVCODE     = 193;
   21           DVCODE     = 224;
   22           ERRLEN     = 128;
   23           GCDEST     = 2222;
   24           GCSOUR     = 1111;
   25           TPDUSZPL   = 1;
   26           TPDUSZPC   = 192;
   27           THPC       = 137;
   28           GCTPDU     = 128;
   29           GCTHR1     = 1;
   30           GCTHR2     = 2;
   31           GCTHR3     = 3;
   32           GCTHR4     = 4;
   33           THPL       = 12;
   34           TRDLPL     = 8;
   35           TRDLPC     = 136;
   36           GCTRD1     = 7;
   37           GCTRD2     = 8;
   38           GCTRD3     = 9;
   39           GCTRD4     = 10;
   40           NREOT      = 99;
   41           GCRESO     = 11;
   42           UNDEF      = 99;
   43
```

```
   44
   45   type                          { GLOBAL TYPES }
   46           std = record          { TPDUs }
   47                   fixp : packed array [1..FIXPLEN] of char;
   48                   oth  : packed array [1..OTHLEN] of char;
   49           end;
   50
   51           pstd = ^std;
   52
   53           contrl = array[0..7] of integer; { format for CRRG$ }
   54
   55           netXreq   = packed array[1..NETREQLEN] of char;
   56
   57           ddd = record
   58                   req : netXreq;
   59                   dum : array[1..DUMMLEN] of integer;
   60           end;
   61
   62           pdd = ^ddd;
   63
   64           tskk = record
   65                   nam : array [1..2] of integer;
   66           end;
   67
   68           rbuff = record { format for RCVD$ }
   69                   nam : tskk;
   70                   req : netXreq;
   71                   dum : array [1..DUMLEN] of integer;
   72           end;
   73
   74           buff = record  { format for SDAT$ }
   75                   req : netXreq;
   76                   dum : array [1..DUMLEN] of integer;
   77           end;
   78
   79           rddd = record
   80                   ok : array[1..2] of integer;
   81                   req : netXreq;
   82                   dum : array[1..DUMMLEN] of integer;
   83           end;
   84
   85           rpdd = ^rddd;
   86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed.array[1..2] of char;
 89                  rvbadd : rstd;
 90                  rsiz : integer;
 91                  rresid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : rstd;
101                  siz : integer;
102                  resid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : rdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFHC); { states of TPRENTA }
110
111          orXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmgs = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..21 of char;
```

```
130
131         errr = integer;
```

```
46
47
48  procedure initial(var id : integer);
49
50  var
51        control  : contrl;
52        ids      : idss;
53
54  procedure crrs(var control : contrl; var ids : idss); extern (fortran);
55
56  begin
57  { writeln(tty, 'entering initial subroutine');}
58
59    (* create and attach to region with a region name of 'A2' *)
60    (* control[0]  region id returned *)
61    control[1] := REGSZ;          (* size of the region in 32-word blocks also returned  *)
62    control[2] := 2880;           (* region name (2 words in Radix-50 format)*)
63    control[3] := 0;              (*  or 0 for no-name, Name is 'A2'          *)
64    control[4] := 11414;          (* Name of the partition that contains the *)
65    control[5] := 0;              (* region.  (2 word in Radix-50 format     *)
66                                  (* presently it is GEN *)
67                                  (* or 0 for the partition in which the task is
68                                     running *)
69    control[6] := 35;             (* Region status word (see page 3-14) also returned *)
70                                  (* bits set:
71                                           RS.ATT = 32
72                                           RS.WRT = 2
73                                           RS.RED = 1                        *)
74
75    control[7] := 0;              (* Region protection code *)
76                                  (* everything is permitted see p 3-13 *)
77    crrs(control, ids);
78  { writeln(tty, 'id assigned to the created region = ', control[0]);
79    writeln(tty, 'size in 32-W blocks of the attached region = ', control[1]);
80    writeln(tty, 'region status word = ', control[6]);
81    writeln(tty, 'ids = ', ids);}
82    if ids < 0 then
83    begin
84       writeln(tty, 'error on crrs,  error code = ', ids);
85       halt
86    end;
87
88    id := control[0] (* stick region id in appropriate return field *)
89
```

```
  90   { writeln(ttw, 'leaving initial subroutine')}
  91   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      000650
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000542
COMPILATION TIME 19 SECONDS

PAS INITIAL,INITIAL/P42-INITIAL
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         INITBUF
    5
    6      FUNCTION:     To initialize a TPDU with zeros in all of its fields.
    7
    8      USAGE:        procedure  initbuf(var swndno : std);extern;
    9                                .
   10                                .
   11                                .
   12                    initbuf(swndno);
   13
   14      INPUT PARAMETERS:   NONE
   15
   16      OUTPUT PARAMETERS:  swndno  - the record to be filled with zeros in all
   17                                    of its fields (initialization).
   18
   19
   20      MODULES CALLED:     ( '>' indicates a primitive )
   21                          >chr
   22                          >writeln
   23
   24      EXIT CONDITIONS:    Normal condition.
   25
   26      REMARKS:
   27                          The FILLB routine should probably have been used here
   28                          instead of chr.
   29
   30  *****************************************************************************)
   31
   32  const                                   { GLOBAL CONSTANTS }
   33        FIXPLEN = 8;
   34        OTHLEN  = 130;
   35  type                                    { GLOBAL TYPE }
   36        std = record
   37                fixp : packed array [1..FIXPLEN] of char;
   38                oth : packed array [1..OTHLEN] of char;
   39        end;
   40
   41
   42
   43  procedure  initbuf(var swndno : std);
```

```
   44
   45   var     i   : integer;
   46           j   : integer;
   47
   48   begin
   49   { writeln(tty, 'entering initbuf routine');}
   50
   51     for i := 1 to FIXPLEN do
   52       swndno.fixp[i] := chr(0);
   53     for j := 1 to OTHLEN do
   54       swndno.oth[j] := chr(0)
   55
   56   { writeln(tty, 'leaving initbuf routine');}
   57   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000274
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000526
COMPILATION TIME 6 SECONDS

PAS INITBUF,INITBUF/P42-INITBUF
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4       NAME:       INSPTPDU
    5
    6       FUNCTION:   To inspect a TPDU for erroneous parameter
    7                   codes or parameter values and returns the length of
    8                   the erroneous packet if an error is detected and sets
    9                   valid to FALSE, otherwise sets valid to TRUE.
   10
   11       USAGE:      procedure insptpdu(ptrwnd : pstd;
   12                                      var valid : boolean;
   13                                      var rjcse : integer;
   14                                      var err : errr); extern;
   15                                .
   16                                .
   17                                .
   18                   insptpdu(ptrwnd, valid, rjcse, err);
   19
   20       INPUT PARAMETERS:   ptrwnd  - the pointer to the TPDU to inspect.
   21
   22       OUTPUT PARAMETERS:  valid   - set to FALSE if error is detected and
   23                                     to TRUE otherwise.
   24                           rjcse   - the detected cause for rejection of TPDU.
   25                           err     - the length of the erroneous string
   26                                     which is sent back.
   27
   28       MODULES CALLED:     (The symbol '>' designates a primitive)
   29
   30                           >ord
   31                           >writeln
   32                           val
   33
   34
   35       EXIT CONDITIONS:
   36                           If an error is detected in the received buffer,
   37                           the length of the erroneous string
   38                           is returned with valid = FALSE
   39                           otherwise valid is set to TRUE.
   40
   41       REMARKS:
   42                           Length returned by this routine is up to and
   43                           including the first erroneous parameter or STRLEN
```

```
   44                          in case of successful inspection.
   45
   46                          At present, the inspection performed is not very
   47                          sophisticated since it is basically done by comparing
   48                          the different fields of a particullar TPDU with some
   49                          fixed global constants which represent the only value
   50                          acceptable.  For our purposes this method has proven to
   51                          be sufficient.
   52
   53   ************************************************************************************)
   54
   55   {$I+TPRENTA.DEF}
```

```
 1
 2   const                          { GLOBAL CONSTANTS }
 3           FIXPLEN    = 8;
 4           OTHLEN     = 130;
 5           STRLEN     = 118;
 6           NETREQLEN  = 6;
 7           CRCODE     = 224;
 8           CCCODE     = 208;
 9           DTCODE     = 240;
10           DRCODE     = 128;
11           NDRCODE    = 120;
12           ERCODE     = 112;
13           APRNO      = 7;
14           WNDSZ      = 3;
15           REGSZ      = 21;  { NOPTR times WNDSZ }
16           NOPTR      = 7;
17           DUMLEN     = 10;
18           DUMMLEN    = 5;
19           NODATFLG   = 20;
20           IVCODE     = 193;
21           DVCODE     = 224;
22           ERRLEN     = 128;
23           GCDEST     = 2222;
24           GCSOUR     = 1111;
25           TPDUSZPL   = 1;
26           TPDUSZPC   = 192;
27           THPC       = 137;
28           GCTPDU     = 128;
29           GCTHR1     = 1;
30           GCTHR2     = 2;
31           GCTHR3     = 3;
32           GCTHR4     = 4;
33           THPL       = 12;
34           TRDLPL     = 8;
35           TRDLPC     = 136;
36           GCTRD1     = 7;
37           GCTRD2     = 8;
38           GCTRD3     = 9;
39           GCTRD4     = 10;
40           NREOT      = 99;
41           GCRESO     = 11;
42           UNDEF      = 99;
43
```

```
44
45   type                          { GLOBAL TYPES }
46          std = record           { TPDUs }
47                  fixp : packed array [1..FIXPLEN] of char;
48                  oth  : packed array [1..OTHLEN] of char;
49          end;
50
51          pstd = ^std;
52
53          contrl = array[0..7] of integer; { format for CRRG$ }
54
55          netXreq   = packed array[1..NETREQLEN] of char;
56
57          ddd = record
58                  req : netXreq;
59                  dum : array[1..DUMMLEN] of integer;
60          end;
61
62          pdd = ^ddd;
63
64          tskk = record
65                  nam : array [1..2] of integer;
66          end;
67
68          rbuff = record { format for RCVD$ }
69                  nam : tskk;
70                  req : netXreq;
71                  dum : array [1..DUMLEN] of integer;
72          end;
73
74          buff = record  { format for SDAT$ }
75                  req : netXreq;
76                  dum : array [1..DUMLEN] of integer;
77          end;
78
79          rddd = record
80                  ok : array[1..2] of integer;
81                  req : netXreq;
82                  dum : array[1..DUMMLEN] of integer;
83          end;
84
85          rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : rstd;
 90                  rsiz : integer;
 91                  rregid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : rstd;
101                  siz : integer;
102                  regid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmgs = integer;
124
125          tnit = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
130
131         errr = integer;
```

```
56
57
58  procedure insptpdu(ptrwnd : pstd;
59                     var valid : boolean;
60                     var rjcse : integer;
61                     var err : errr);
62  var
63        i : integer;
64
65  function val(c1 : char; c2 : char) : integer ; extern;
66
67  begin
68
69  { writeln(tty, 'entering insptpdu subroutine');}
70
71    valid := true;  (* valid until proven otherwise *)
72    err := ERRLEN;  (* length of the error string *)
73    rjcse := 0;     (* no cause specified *)
74
75    CASE ord(ptrwnd^.fixp[1]) OF  (* select the proper TPDU type *)
76
77      CRCODE:
78      begin
79        (* LI field ????????? *)
80
81        (* DESTINATION ADDRESS *)
82        if (val(ptrwnd^.fixp[3], ptrwnd^.fixp[4]) <> GCSOUR) then
83        begin
84          err := 5;
85          rjcse := 3;
86          valid := false
87        end
88        else (* SOURCE ADDRESS *)
89        if (val(ptrwnd^.fixp[5], ptrwnd^.fixp[6]) <> GCDEST) then
90        begin
91          err := 7;
92          rjcse := 3;
93          valid := false
94        end
95        else (* TPDUSZPL *)
96        if (ord(ptrwnd^.oth[7]) <> TPDUSZPL) then
97        begin
98          err := 16;
99          rjcse := 3;
```

```
100           valid := false
101         end
102         else (* TPDUSZPC *)
103         if (ord(ptrwnd^.oth[8]) <> TPDUSZPC) then
104         begin
105           err := 17;
106           rjcse := 1;
107           valid := false
108         end
109         else (* THPC *)
110         if (ord(ptrwnd^.oth[9]) <> THPC) then
111         begin
112           err := 18;
113           rjcse := 1;
114           valid := false
115         end
116         else (* TPDU SIZE *)
117         if (ord(ptrwnd^.oth[10]) <> GCTPDU) then
118         begin
119           err := 19;
120           rjcse := 3;
121           valid := false
122         end
123         else (* THROUGHPUT 1 *)
124         if (val(ptrwnd^.oth[11], ptrwnd^.oth[12]) <> GCTHR1) then
125         begin
126           err := 21;
127           rjcse := 3;
128           valid := false
129         end
130         else (* THROUGHPUT 2 *)
131         if (val(ptrwnd^.oth[13], ptrwnd^.oth[14]) <> GCTHR2) then
132         begin
133           err := 23;
134           rjcse := 3;
135           valid := false
136         end
137         else (* THROUGHPUT 3 *)
138         if (val(ptrwnd^.oth[15], ptrwnd^.oth[16]) <> GCTHR3) then
139         begin
140           err := 25;
141           rjcse := 3;
142           valid := false
```

```
143         end
144         else (* THROUGHPUT 4 *)
145         if (val(ptrwnd^.oth[17], ptrwnd^.oth[18]) <> GCTHR4) then
146         begin
147           err := 27;
148           rjcse := 3;
149           valid := false
150         end
151         else (* THPL *)
152         if (ord(ptrwnd^.oth[23]) <> THPL) then
153         begin
154           err := 32;
155           rjcse := 3;
156           valid := false
157         end
158         else (* TRDLPL *)
159         if (ord(ptrwnd^.oth[24]) <> TRDLPL) then
160         begin
161           err := 33;
162           rjcse := 3;
163           valid := false
164         end
165         else (* TRDLPC *)
166         if (ord(ptrwnd^.oth[25]) <> TRDLPC) then
167         begin
168           err := 34;
169           rjcse := 1;
170           valid := false
171         end
172         else (* TRANSIT DELAY 1 *)
173         if (val(ptrwnd^.oth[27], ptrwnd^.oth[28]) <> GCTRD1) then
174         begin
175           err := 37;
176           rjcse := 3;
177           valid := false
178         end
179         else (* TRANSIT DELAY 2 *)
180         if (val(ptrwnd^.oth[29], ptrwnd^.oth[30]) <> GCTRD2) then
181         begin
182           err := 39;
183           rjcse := 3;
184           valid := false
185         end
```

```
186         else (* TRANSIT DELAY 3 *)
187         if (val(ptrwnd^.oth[31], ptrwnd^.oth[32]) <> GCTRD3) then
188         begin
189           err := 41;
190           rjcse := 3;
191           valid := false
192         end
193         else (* TRANSIT DELAY 4 *)
194         if (val(ptrwnd^.oth[33], ptrwnd^.oth[34]) <> GCTRD4) then
195         begin
196           err := 43;
197           rjcse := 3;
198           valid := false
199         end
200       end;
201
202       CCCODE:
203       begin
204         (* LI field ????????? *)
205
206         (* SOURCE ADDRESS *)
207         if (val(ptrwnd^.fixp[5], ptrwnd^.fixp[6]) <> GCDEST) then
208         begin
209           err := 7;
210           rjcse := 3;
211           valid := false
212         end
213         else (* TPDUSZPL *)
214         if (ord(ptrwnd^.oth[7]) <> TPDUSZPL) then
215         begin
216           err := 16;
217           rjcse := 3;
218           valid := false
219         end
220         else (* TPDUSZPC *)
221         if (ord(ptrwnd^.oth[8]) <> TPDUSZPC) then
222         begin
223           err := 17;
224           rjcse := 1;
225           valid := false
226         end
227         else (* THPC *)
228         if (ord(ptrwnd^.oth[9]) <> THPC) then
```

```
229         begin
230           err := 18;
231           rjcse := 1;
232           valid := false
233         end
234         else (* TPDU SIZE *)
235         if (ord(ptrwnd^.oth[10]) <> GCTPDU) then
236         begin
237           err := 19;
238           rjcse := 3;
239           valid := false
240         end
241         else (* THROUGHPUT 1 *)
242         if (val(ptrwnd^.oth[11], ptrwnd^.oth[12]) <> GCTHR1) then
243         begin
244           err := 21;
245           rjcse := 3;
246           valid := false
247         end
248         else (* THROUGHPUT 2 *)
249         if (val(ptrwnd^.oth[13], ptrwnd^.oth[14]) <> GCTHR2) then
250         begin
251           err := 23;
252           rjcse := 3;
253           valid := false
254         end
255         else (* THROUGHPUT 3 *)
256         if (val(ptrwnd^.oth[15], ptrwnd^.oth[16]) <> GCTHR3) then
257         begin
258           err := 25;
259           rjcse := 3;
260           valid := false
261         end
262         else (* THROUGHPUT 4 *)
263         if (val(ptrwnd^.oth[17], ptrwnd^.oth[18]) <> GCTHR4) then
264         begin
265           err := 27;
266           rjcse := 3;
267           valid := false
268         end
269         else (* THPL *)
270         if (ord(ptrwnd^.oth[23]) <> THPL) then
271         begin
```

```
272           err := 32;
273           rjcse := 3;
274           valid := false
275         end
276         else (* TRDLPL *)
277         if (ord(ptrwnd^.oth[24]) <> TRDLPL) then
278         begin
279           err := 33;
280           rjcse := 3;
281           valid := false
282         end
283         else (* TRDLPC *)
284         if (ord(ptrwnd^.oth[25]) <> TRDLPC) then
285         begin
286           err := 34;
287           rjcse := 1;
288           valid := false
289         end
290         else (* TRANSIT DELAY 1 *)
291         writeln(tty, 'tr1 - ', val(ptrwnd^.oth[27], ptrwnd^.oth[28]));
292         if (val(ptrwnd^.oth[27], ptrwnd^.oth[28]) <> GCTRD1) then
293         begin
294           err := 37;
295           rjcse := 3;
296           valid := false
297         end
298         else (* TRANSIT DELAY 2 *)
299         if (val(ptrwnd^.oth[29], ptrwnd^.oth[30]) <> GCTRD2) then
300         begin
301           err := 39;
302           rjcse := 3;
303           valid := false
304         end
305         else (* TRANSIT DELAY 3 *)
306         if (val(ptrwnd^.oth[31], ptrwnd^.oth[32]) <> GCTRD3) then
307         begin
308           err := 41;
309           rjcse := 3;
310           valid := false
311         end
312         else (* TRANSIT DELAY 4 *)
313         if (val(ptrwnd^.oth[33], ptrwnd^.oth[34]) <> GCTRD4) then
314         begin
```

```
315           err := 43;
316           rjcse := 3;
317           valid := false
318         end
319       end;
320
321       DTCODE:
322       begin
323         (* LI field ????????? *)
324
325         if (ord(ptrwnd^.fixp[4]) <> NREOT) then
326         begin
327           err := 5;
328           rjcse := 3;
329           valid := false
330         end
331         else (* check if length of USER DATA is not too long *)
332         if (ord(ptrwnd^.uth[1]) > STRLEN) then
333         begin
334           err := 10;
335           rjcse := 3;
336           valid := false
337         end
338       end;
339
340       DRCODE:
341       begin
342         (* LI field ????????? *)
343
344         (* DESTINATION ADDRESS *)
345         if (val(ptrwnd^.fixp[3], ptrwnd^.fixp[4]) <> GCSOUR) then
346         begin
347           err := 5;
348           rjcse := 3;
349           valid := false
350         end
351         else (* SOURCE ADDRESS *)
352         if (val(ptrwnd^.fixp[5], ptrwnd^.fixp[6]) <> GCDEST) then
353         begin
354           err := 7;
355           rjcse := 3;
356           valid := false
357         end
```

```
358         else (* REASON *)
359         if (ord(ptrwnd^.fixp[8]) <> GCRESO) then
360         begin
361           err := 9;
362           rjcse := 3;
363           valid := false
364         end
365         else (* legal DVL *)
366         if (ord(ptrwnd^.oth[1]) <> STRLEN) then
367         begin
368           err := 10;
369           rjcse := 3;
370           valid := false
371         end
372         else (* DVCODE *)
373         if (ord(ptrwnd^.oth[2]) <> DVCODE) then
374         begin
375           err := 11;
376           rjcse := 1;
377           valid := false
378         end
379       end;
380
381       OTHERS:   (* all other codes *)
382       begin
383         err := 2; (* error in code *)
384         rjcse := 2;
385         valid := false
386       end
387
388     end;    (* CASE *)
389
390   { writeln(tty, 'leaving insptpdu subroutine')}
391
392   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      012600
OUTERHOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 110 SECONDS

PAS INSPTPDU,INSPTPDU/P42=INSPTPDU
```

```
1   (*$m-*)
2   (*****************************************************************************
3
4      NAME:          NTAINIT
5
6      FUNCTION:      This module is used to create a dynamic region for
7                     task NTRNSDA by issuing a CRRG$ memory management directive.
8
9      USAGE:         procedure ntainit(var id : integer); extern;
10                                .
11                                .
12                                .
13                    ntainit(id);
14
15     INPUT PARAMETERS:    NONE
16
17     OUTPUT PARAMETERS:   id       - the region id.
18
19     MODULES CALLED:      (The symbol '>' designates a primitive)
20
21                          >halt
22                          >writeln
23                          crrg
24
25
26     EXIT CONDITIONS:
27                          If there is an error detected in calling CRRG$, then
28                          an error message identifying the error and the error
29                          code is printed onto standard I/O and the processor
30                          is then halted by using the 'halt' primitive.
31
32                          The return code of the CRRG$ routine is set to greater
33                          or equal to 0 (zero) in case if successful completion
34                          and to less than 0 in case of unsuccessful completion.
35
36                          For more information on the use of CRRG$ and of its
37                          return codes, see pages 6-36, 6-38 of the RSX-11M V3.2
38                          Executive Reference Manual.
39
40     REMARKS:             At the moment, there is no protection on the region.
41                          One can easily remedy this by giving an appropriate
42                          value to the region protection word.
43
```

```
44   ********************************************************************************)
45
46   (*$I+NTRNSDA.DEF*)
```

```
 1
 2  const
 3          FIXPLEN    = 8;
 4          OTHLEN     = 130;
 5          NETREQLEN  = 6;
 6          APRNO      = 7;
 7          WNDSZ      = 3;
 8          REGSZ      = 21;  { NOPTR times WNDSZ }
 9          NOPTR      = 7;
10          DUMLEN     = 10;
11          DUMMLEN    = 5;
12          NODATFLG   = 21;
13          TERMAFLAG  = 42;
14          TERMBFLAG  = 40;
15          TRNSDAFLAG = 43;
16          TRNSDBFLAG = 41;
17          CRCODE     = 224;
18          CCCODE     = 208;
19          DRCODE     = 128;
20          DTCODE     = 240;
21          ERCODE     = 112;
22
23  type
24          std = record
25                  fixp : packed array [1..FIXPLEN] of char;
26                  oth  : packed array [1..OTHLEN] of char;
27          end;
28
29          pstd = ^std;
30
31          contrl = array[0..7] of integer; { format for CRRG$ Executive Directive }
32
33          netXreq   = packed array[1..NETREQLEN] of char;
34
35          ddd = record
36                  req : netXreq;
37                  dum : array[1..DUMMLEN] of integer;
38          end;
39
40          pdd = ^ddd;
41
42          tskk = record
43                  num : array [1..2] of integer;
```

```
44          end;
45
46          rbuff = record
47                  nam : tskk;
48                  req : netXreq;
49                  dum : array [1..DUMLEN] of integer;
50          end;
51
52          buff = record
53                  req : netXreq;
54                  dum : array [1..DUMLEN] of integer;
55          end;
56
57          rddd = record
58                  ok : array [1..2] of integer;
59                  req : netXreq;
60                  dum : array[1..DUMMLEN] of integer;
61          end;
62
63          rpdd = ^rddd;
64
65          rwdbb = record
66                  rapr : packed array[1..2] of char;
67                  rvbadd : pstd;
68                  rsiz : integer;
69                  rregid : integer;
70                  roffs : integer;
71                  rlen : integer;
72                  rwstats : integer;
73                  rsrbuff : rpdd;
74          end;
75
76          wdbb = record
77                  apr : packed array[1..2] of char;
78                  vbadd : pstd;
79                  siz : integer;
80                  regid : integer;
81                  offs : integer;
82                  len : integer;
83                  wstats : integer;
84                  srbuff : pdd;
85          end;
86
```

```
87          state     = (one, two, three, four);
88
89          idss = integer;
90
91          efnn = integer;
92
93          tass = integer;
94
95          tntt = integer;
96
97          wndnoo = 1..NOPTR;
98
```

```
47
48
49  procedure ntainit(var id : integer);
50
51  var
52          control  : contrl;
53          ids      : idss;
54
55  procedure crrs(var control : contrl; var ids : idss); extern (fortran);
56
57  begin
58  { writeln(tty, 'entering ntainit subroutine');}
59
60    (* create and attach to region with region name 'A3' *)
61    (* control[0]  region id returned *)
62    control[1] := REGSZ;          (* size of the region in 32-word blocks also returned  *)
63    control[2] := 2920;           (* region name (2 words in Radix-50 format)*)
64    control[3] := 0;              (*  or 0 for no-name.  Name is 'A3'          *)
65    control[4] := 11414;          (* Name of the partition that contains the *)
66    control[5] := 0;              (* region.  (2 word in Radix-50 format      *)
67                                  (* presently it is GEN *)
68                                  (* or 0 for the partition in which the task is
69                                     running *)
70    control[6] := 35;             (* Region status word (see page 3-14) also returned *)
71                                  (* bits set:
72                                           RS.ATT = 32
73                                           RS.WRT = 2
74                                           RS.RED = 1                    *)
75
76    control[7] := 0;              (* Region protection code *)
77                                  (* everything is permitted see p 3-13 *)
78    crrs(control, ids);
79  { writeln(tty, 'id assigned to the created region = ', control[0]);
80    writeln(tty, 'size in 32-W blocks of the attached region = ', control[1]);
81    writeln(tty, 'region status word = ', control[6]);
82    writeln(tty, 'ids = ', ids);}
83    if ids < 0 then
84    begin
85      writeln(tty, 'error on crrg,  error code = ', ids);
86      halt
87    end;
88
89    id := control[0]; (* stick region id in appropriate return field *)
90
```

```
PASCAL, PDP-11  VERSION  6.07        1908-08-04      04:49:31          PAGE   7
NTAINIT.PAS                            NTAINIT

   91    { writeln(tty, 'leaving ntainit subroutine')}
   92    end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE       000656
OUTERHOST DATA SIZE      000002
RESERVED STACK & HEAP    000542
COMPILATION TIME 16 SECONDS

PAS NTAINIT,NTAINIT/P42=NTAINIT
```

```
    1  (*$m-*)
    2  (*****************************************************************************
    3
    4     NAME:       NTAREC
    5
    6     FUNCTION:   To wait until some information is sent to the NTRNSDA task
    7                 via send data or send by reference, and receives it via
    8                 receive data or receive by reference.
    9
   10     USAGE:      procedure ntarec(var ptrwnd : pstd;
   11                                  var nXreu : netXreq;
   12                                  var queue : boolean;
   13                                  var messXtprenta : boolean); extern;
   14
   15
   16
   17                ntarec(ptrwnd, nXreq, queue, messXtprenta);
   18
   19
   20     INPUT PARAMETERS     NONE
   21
   22     OUTPUT PARAMETERS:   ptrwnd  - the pointer to receive by reference buffer.
   23                          nXreu   - receive data string or receive by reference
   24                                    string.
   25                          queue   - true if a receive data was performed
   26                                    false if a receive by reference was done.
   27                          messXtprenta- true if data was sent by TPRENTA task
   28                                        false if data was sent by NTRNSDB task.
   29
   30     MODULES CALLED:      (The symbol '>' designates a primitive)
   31
   32                          >chr
   33                          >halt
   34                          >writeln
   35                          buffwait
   36                          craw
   37                          dirg
   38                          mark
   39                          receiv
   40                          rref
   41
   42     EXIT CONDITIONS:
   43                          If there are any errors detected in calling CRAW$, DTRG$,
```

```
                        MARK$, RECEIV$ or RREF$, then an error message identifying
                        the error and error code is printed onto standard I/O and
                        the processor is then halted by usint.the 'halt' primitive.

                        The return code of all these system directives, is set to
                        greater or equal to 0(zero) in case of successful completion
                        and to less than 0 in case of unsuccessful completion.

                        For more information on the use of these system routines
                        and their return codes, see the RSX-11M V3.2 Executive
                        Reference Manual.

    REMARKS:            When there is a receive by reference, the newly attached region
                        is detached in order to prevent depletion of the memory pool.
                        See page 6-108 of Executive Reference Manual for more details.

*****************************************************************************)

{$I+TPRENTA.DEF}
```

```
 1
 2  const                          { GLOBAL CONSTANTS }
 3          FIXPLEN   = 8;
 4          OTHLEN    = 130;
 5          STRLEN    = 118;
 6          NETREQLEN = 6;
 7          CRCODE    = 224;
 8          CCCODE    = 208;
 9          DTCODE    = 240;
10          DRCODE    = 128;
11          NDRCODE   = 120;
12          ERCODE    = 112;
13          APRNO     = 7;
14          WNDSZ     = 3;
15          REGSZ     = 21;  { NOPTR times WNDSZ }
16          NOPTR     = 7;
17          DUMLEN    = 10;
18          DUMMLEN   = 5;
19          NODATFLG  = 20;
20          IVCODE    = 193;
21          DVCODE    = 224;
22          ERRLEN    = 128;
23          GCDEST    = 2222;
24          GCSOUR    = 1111;
25          TPDUSZPL  = 1;
26          TPDUSZPC  = 192;
27          THPC      = 137;
28          GCTPDU    = 128;
29          GCTHR1    = 1;
30          GCTHR2    = 2;
31          GCTHR3    = 3;
32          GCTHR4    = 4;
33          THPL      = 12;
34          TRDLFL    = 8;
35          TRDLPC    = 136;
36          GCTRD1    = 7;
37          GCTRD2    = 8;
38          GCTRD3    = 9;
39          GCTRD4    = 10;
40          NREOT     = 99;
41          GCRESO    = 11;
42          UNDEF     = 99;
43
```

```
 44
 45   type                          { GLOBAL TYPES }
 46        std = record             { TPDUs }
 47                fixp : packed array [1..FIXPLEN] of char;
 48                oth  : packed array [1..OTHLEN] of char;
 49        end;
 50
 51        pstd = ^std;
 52
 53        contrl = array[0..7] of integer; { format for CRRG$ }
 54
 55        netXreq   = packed array[1..NETREQLEN] of char;
 56
 57        ddd = record
 58                req : netXreq;
 59                dum : array[1..DUMMLEN] of integer;
 60        end;
 61
 62        pdd = ^ddd;
 63
 64        tskk = record
 65                nam : array [1..2] of integer;
 66        end;
 67
 68        rbuff = record { format for RCVD$ }
 69                nam : tskk;
 70                req : netXreq;
 71                dum : array [1..DUMLEN] of integer;
 72        end;
 73
 74        buff = record  { format for SDAT$ }
 75                req : netXreq;
 76                dum : array [1..DUMLEN] of integer;
 77        end;
 78
 79        rddd = record
 80                ok : array[1..2] of integer;
 81                req : netXreq;
 82                dum : array[1..DUMMLEN] of integer;
 83        end;
 84
 85        rpdd = ^rddd;
 86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rresid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  resid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmsg = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
130
131             errr = integer;
```

```
63
64
65  procedure buffwait(efn : efnn); extern;
66
67  procedure ntarec(var ptrwnd : pstd;
68                   var nXreq : netXreq;
69                   var queue : boolean;
70                   var messXtprenta : boolean);
71
72  var
73          dummy   : efnn;
74          efn     : efnn;
75          gotit   : boolean;
76          ids     : idss;
77          irdb    : contrl;
78          isrb    : rddd;
79          rbuf    : rbuff;
80          rwdb    : rwdbb;
81          rrwdb   : rwdbb;
82          tmg     : tmgg;
83          tnt     : tntt;
84          tsk     : tskk;
85
86  procedure craw(var rwdb : rwdbb; var ids : idss); extern(fortran);
87
88  procedure dtrg(var irdb : contrl; var ids : idss); extern(fortran);
89
90  procedure mark(var efn : efnn;
91                 var tmg : tmgg;
92                 var tnt : tntt;
93                 var ids : idss); extern(fortran);
94
95  procedure receiv(var tsk : tskk;
96                   var rbuf : rbuff;
97                   var dummy : efnn;
98                   var ids : idss); extern(fortran);
99
100 procedure rref(var rrwdb : rwdbb;
101                var isrb : rddd;
102                var ids : idss); extern(fortran);
103
104 begin
105 { writeln(tty, ' entering ntarec procedure');}
106
```

```
107     messXtprenta := false;   (* initialize booleans *)
108     queue := false;
109     gotit := false;
110     efn := NODATFLG;         (* initialize local event flag *)
111     while not gotit do       (* do until something is received *)
112     begin
113
114       (* receive the data *)
115       receiv(tsk, rbuf, dummy, ids);
116   {   writeln(tty, 'receiv ids = ', ids);}
117       if ids <> -8 then
118       begin
119         if ids < 0 then
120         begin
121           writeln(tty, 'error in receiving data using "receiv" error code = ', ids);
122           halt
123         end
124         else
125         begin
126           nXreq := rbuf.req;
127           gotit := true;
128           queue := true;
129
130           (* check if message was sent by tprenta *)
131           if ((rbuf.nam.nam[1] = 32658) and (rbuf.nam.nam[2] = 8561)) then
132             messXtprenta := true
133         end
134       end
135       else    (* receive by reference *)
136       begin
137         (* prepare to receive by reference *)
138
139         (* window definition block *)
140         rrwdb.rwstats := 0;          (* window status word *)
141                                      (* bits set : NONE    *)
142         rref(rrwdb, isrb, ids);
143   {     writeln(tty, 'rref ids = ', ids);}
144         if ids <> -8 then
145         begin
146           if ids < 0 then
147           begin
148             writeln(tty, ' error in using rref,  error code = ',ids);
149             halt
```

```
150           end
151           else
152           begin  (* successful receive by reference *)
153     {       writeln(tty, 'ptr to buff=',ord(rrwdb.rsrbuff));
154             writeln(tty, 'region id (pointer to attachment description=',ord(rrwdb.rregid));
155             writeln(tty, 'offset word specified by sender task=',rrwdb.roffs);
156             writeln(tty, 'length word specified by sender task=',rrwdb.rlen);
157             writeln(tty, 'window status word =', rrwdb.rwstats);
158             writeln(tty, 'virtual base address =', ord(rrwdb.rvbadd));
159             writeln(tty, 'ids = ', ids);}
160
161             gotit := true;
162
163         (* create receive window into the newly attached region *)
164             rwdb.rapr[2] := chr(APRNO);    (* Active page register 7 *)
165             rwdb.rsiz := WNDSZ;            (* the size of the window in 32-word blocks *)
166             rwdb.rregid := rrwdb.rregid;   (* region id *)
167                                            (* or 0 for task region *)
168                                            (* specified only if WS.MAP is set *)
169             rwdb.roffs := rrwdb.roffs;         (* offset in region (32W blocks) at which
170                                                the window is to start mapping *)
171                                            (* specified only if WS.MAP is set *)
172             rwdb.rlen := rrwdb.rlen;       (* length to map (32W blocks) *)
173                                            (* or 0 if the length is to default to either
174                                               the size of the window or the space remaining
175                                               in the region *)
176                                            (* specified only if WS.MAP is set *)
177             rwdb.rwstats := 386;           (* window status word *)
178                                            (* bits set:
179                                                WS.64B = 256
180                                                WS.MAP = 128
181                                                WS.WRT = 2                *)
182             crew(rwdb, ids);
183     {       writeln(tty, 'window created ids =', ids);
184             writeln(tty, 'Id assigned to the window = ',ord(rwdb.rapr[1]));
185             writeln(tty, 'virtual base address = ', ord(rwdb.rvbadd));
186             writeln(tty, 'length actually mapped in 32-word blocks - ', rwdb.rlen);
187             writeln(tty, 'window status word = ', rwdb.rwstats);}
188             if ids < 0 then
189             begin
190                writeln(tty, ' error in using crew,  error code - ',ids);
191                halt
192             end;
```

```
193
194
195           (* put receive packet in returned buffer *)
196           ptrwnd^.fixp := rwdb.rvbadd^.fixp;
197           ptrwnd^.oth := rwdb.rvbadd^.oth;
198
199           (* put receive by reference control buffer into string field *)
200           nXreq := isrb.req;
201
202           (* detach from newly attached region *)
203           irdb[0] := rrwdb.rregid; (* region id *)
204           irdb[6] := 0;  (* no bits set *)
205           dtrg(irdb, ids);
206           if ids < 0 then
207           begin
208             writeln(tty, 'error trying to detach from newly attached region');
209             halt
210           end;
211
212           (* check if message was sent by terenta *)
213           if ((rrwdb.rsrbuff^.uk[1] = 32658) and (rrwdb.rsrbuff^.uk[2] = 8561)) then
214           begin
215   {         writeln(tty, 'message received from terenta');}
216             messXterenta := true
217           end
218         end
219       end
220       else
221       begin          (* wait for a while and try again *)
222         tms := 10; (* wait 10  *)
223         tnt := 2; (* seconds *)
224         mark(efn, tms, tnt, ids);
225         buffwait(efn)
226       end
227     end
228   end
229
230 { writeln(tty, ' leaving  ntarec procedure');}
231 end.

NO ERRORS DETECTED
TOTAL PROGRAM SIZE      002666
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000702
```

COMPILATION TIME: 5/ SECONDS

FAS NTAREC,NTAREC/P42=NTAREC

```
 1  (*****************************************************************************
 2
 3     NAME:        NTRNSDA (MAIN)
 4
 5     FUNCTION:    Network transducer for side 'A'.
 6
 7     USAGE:       from MCR : 'RUN NTRNSDA /TASK=TRNSDA /INC=100.'
 8
 9     INPUT PARAMETERS:    NONE
10
11     OUTPUT PARAMETERS:   NONE
12
13     MODULES CALLED:      (The symbol '>' designates a primitive)
14
15                          >chr
16                          >new
17                          >ord
18                          >write
19                          >writeln
20                          buffclear
21                          buffgu
22                          buffwait
23                          ntainit
24                          ntarec
25                          sdtprenta
26                          wuab
27                          wmess
28
29     EXIT CONDITIONS:
30                          This program has been designed to run indefinitely until
31                          the task into which it runs is specifically aborted with
32                          the MCR command 'ABORT TRNSDA'.
33
34                          If any errors are detected by one of the routines called,
35                          then the routine itself will print an error message and
36                          halt the processor by using the 'halt' primitive.
37
38     REMARKS:
39                          See Ref. (22) for more details.
40
41  *****************************************************************************)
42
43  program ntrnsda(tty);
```

```
44
45   {$I+NTRNSDA.DEF}
```

```
    1
    2   const
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           NETREQLEN  = 6;
    6           APRNO      = 7;
    7           WNDSZ      = 3;
    8           REGSZ      = 21;  { NOPTR times WNDSZ }
    9           NOPTR      = 7;
   10           DUMLEN     = 10;
   11           DUMMLEN    = 5;
   12           NODATFLG   = 21;
   13           TERMAFLAG  = 42;
   14           TERMBFLAG  = 40;
   15           TRNSDAFLAG = 43;
   16           TRNSDBFLAG = 41;
   17           CRCODE     = 224;
   18           CCCODE     = 208;
   19           DRCODE     = 128;
   20           DTCODE     = 240;
   21           ERCODE     = 112;
   22
   23   type
   24           std = record
   25                   fixp : packed array [1..FIXPLEN] of char;
   26                   oth  : packed array [1..OTHLEN] of char;
   27           end;
   28
   29           pstd = ^std;
   30
   31           contrl = array[0..7] of integer; { format for CRRG$ Executive Directive }
   32
   33           netXreq   = packed array[1..NETREQLEN] of char;
   34
   35           ddd = record
   36                   req : netXreq;
   37                   dum : array[1..DUMMLEN] of integer;
   38           end;
   39
   40           pdd = ^ddd;
   41
   42           tskk = record
   43                   nam : array [1..2] of integer;
```

```
44          end;
45
46          rbuff = record
47                  nam : tskk;
48                  req : netXreq;
49                  dum : array [1..DUMLEN] of integer;
50          end;
51
52          buff = record
53                  req : netXreq;
54                  dum : array [1..DUMLEN] of integer;
55          end;
56
57          rddd = record
58                  ok : array [1..2] of integer;
59                  req : netXreq;
60                  dum : array[1..DUMMLEN] of integer;
61          end;
62
63          rpdd = ^rddd;
64
65          rwdbb = record
66                  rapr : packed array[1..2] of char;
67                  rvbadd : rstd;
68                  rsiz : integer;
69                  rregid : integer;
70                  roffs : integer;
71                  rlen : integer;
72                  rwstats : integer;
73                  rsrbuff : rpdd;
74          end;
75
76          wdbb = record
77                  apr : packed array[1..2] of char;
78                  vbadd : rstd;
79                  siz : integer;
80                  regid : integer;
81                  offs : integer;
82                  len : integer;
83                  wstats : integer;
84                  srbuff : pdd;
85          end;
86
```

```
87          state     = (one, two, three, four);
88
89          idss = integer;
90
91          efnn = integer;
92
93          tags = integer;
94
95          tntt = integer;
96
97          wndnoo = 1..NOPTR;
98
```

```
46
47
48  var
49          efn             : efnn;
50          id              : integer;
51          ids             : idss;
52          messXtprenta    : boolean;
53          nXreq           : netXreq;
54          ptrwnd          : pstd;
55          queue           : boolean;
56          rdbuf           : netXreq;
57          rrbuf           : netXreq;
58          sdbuf           : buff;
59          srbuf           : ddd;
60          stateXa         : state;
61          tms             : tmsg;
62          tnt             : tntt;
63          wid             : char;
64          wndno           : wndnoo;
65
66  procedure buffclear(efn : efnn); extern;
67
68  procedure buffgo(efn : efnn); extern;
69
70  procedure buffwait(efn : efnn); extern;
71
72  procedure ntainit(var id : integer); extern;
73
74  procedure ntarec(var ptrwnd : pstd;
75                   var nXreq : netXreq;
76                   var queue : boolean;
77                   var messXtprenta : boolean); extern;
78
79  procedure sdtprenta(ptrwnd : pstd;
80                  var wndno : wndnoo;
81                  nXreq : netXreq;
82                  id : integer;
83                  bol : boolean); extern;
84
85  procedure waab(ptrwnd : pstd;
86                  var wndno : wndnoo;
87                  nXreq : netXreq;
88                  id : integer;
89                  bol : boolean); extern;
```

```
 90
 91   procedure wmess(messno : integer;
 92                   stateXa : state;
 93                   ddum : netXreq); extern;
 94
 95   begin
 96     (* for display purposes on VT100 compatible terminals *)
 97   { writeln(tty, chr(27),'[2J',
 98                  chr(27), '[0;0f', chr(27), '[0m',
 99                  chr(27), '[7;24r', chr(27), '[7m', chr(27), '[A');
100     writeln(tty, '*****************************************************************************************');
101     writeln(tty, '*                                                                                       *');
102     writeln(tty, '*                  T A S K   N T R N S D A  activated                                    *');
103     writeln(tty, '*                                                                                       *');
104     writeln(tty, '*****************************************************************************************');
105     writeln(tty, chr(27), '[0m');}
106
107     wndno := 1;             (* initialize sending window *)
108     stateXa := one;         (* initialize state of NTRNSDA *)
109     new(ptrwnd);            (* allocate space for TPDU *)
110     ntainit(id);            (* create dynamic region with name 'A3' *)
111
112     (* make sure the screen is clear *)
113     writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
114     writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
115     writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
116
117     while(1 = 1) do         (* DO forever *)
118     begin
119
120       (* print out the state of the transducer *)
121       write(tty, chr(27), '[7;20f', 'STATE < ');
122       if (ord(stateXa) = 0) then
123         writeln(tty, 'IDLE          >')
124       else if (ord(stateXa) = 1) then
125         writeln(tty, 'OUT_CON_PEND >')
126       else if (ord(stateXa) = 2) then
127         writeln(tty, 'IN_CON_PEND  >')
128       else
129         writeln(tty, 'DATA_TRANSFER>');
130
131       (* receive packet from receive by reference queue or by task queue *)
132       ntarec(ptrwnd, nXreq, queue, messXtorenta);
```

```
133
134       (* print out the type of primitive and /or TPDU received *)
135       if queue then
136       begin
137         writeln(tty, chr(27), '[8;20f', 'RECEIVED : ', nXreq);
138         writeln(tty, chr(27), '[9;20f' , 'TPDU  : NONE          ')
139       end
140       else if not messXtprenta then
141       begin
142         writeln(tty, chr(27), '[8;20f', 'RECEIVED : ', nXreq);
143         write(tty, chr(27), '[9;20f' , 'TPDU  :')
144         if ord(ptrwnd^.fixp[1]) = CRCODE then
145           writeln(tty, ' "CR"')
146         else if ord(ptrwnd^.fixp[1]) = CCCODE then
147           writeln(tty, ' "CC"')
148         else if ord(ptrwnd^.fixp[1]) = DTCODE then
149           writeln(tty, ' "DT"')
150         else if ord(ptrwnd^.fixp[1]) = DRCODE then
151           writeln(tty, ' "DR"')
152         else if ord(ptrwnd^.fixp[1]) = ERCODE then
153           writeln(tty, ' "ER"')
154         else
155           writeln(tty, ' "XX"')
156       end
157       else
158       begin
159         writeln(tty, chr(27), '[8;20f', 'RECEIVED :          ');
160         write(tty, chr(27), '[9;20f' , 'TPDU  :')
161         if ord(ptrwnd^.fixp[1]) = CRCODE then
162           writeln(tty, ' "CR"')
163         else if ord(ptrwnd^.fixp[1]) = CCCODE then
164           writeln(tty, ' "CC"')
165         else if ord(ptrwnd^.fixp[1]) = DTCODE then
166           writeln(tty, ' "DT"')
167         else if ord(ptrwnd^.fixp[1]) = DRCODE then
168           writeln(tty, ' "DR"')
169         else if ord(ptrwnd^.fixp[1]) = ERCODE then
170           writeln(tty, ' "ER"')
171         else
172           writeln(tty, ' "XX"')
173       end;
174       writeln(tty, chr(27), '[23;1f');
175
```

```
176
177       if (messXtprenta) then        (* the incoming message is from TPRENTA *)
178       begin                         (* YES,. *)
179         case stateXa of
180
181
182         (* if in idle state then do the following *)
183
184
185     one :  begin
186              if (queue) then                    (* receive data *)
187              begin
188                if (nXreq = 'ncreq ') then
189                begin
190                  stateXa  := two;                 (* outXconnXpending  *)
191                  wqab(ptrwnd, wndno, 'a     ', id, true)         (* SEND data to NTRNSDB *)
192                end
193                else                          (* write an error message for debug
194                                               purposes       *)
195                  wmess(6, stateXa, nXreq)
196              end
197              else             (* if NOT queue then write for debug puposes *)
198              begin
199                writeln(tty, 'RECEIVE by ref. does not work in IDLE state');
200                writeln(tty, 'only strings such as NCREQ etc are accepted')
201              end
202            end;                                 (* idle *)
203
204         (* if in outXconnXpending state then do the following *)
205
206
207     two :  begin
208              if (queue) then                    (* RECEIVE data *)
209              begin
210                if (nXreq = 'ndreq ') then
211                begin
212                  stateXa := one;                  (* back to idle state *)
213                  wqab(ptrwnd, wndno, 'd     ', id, true)      (* SEND data to NTRNSDB *)
214                end
215                else                             (* for debug purposees *)
216                  wmess(6, stateXa, nXreq)
217              end
218              else                               (* for debug purposes *)
```

```
219             begin
220               writeln(tty, 'RECEIVE by ref. does not work in OUTCONNECT PENDING');
221               writeln(tty, 'state; only strings such as NDREQ etc are accepted')
222             end
223           end;                                  (* outXconnXpending *)
224
225
226         (* if in inXconnXpending state, then do the following *)
227
228
229     three : begin
230               if (queue) then                   (* RECEIVE data *)
231               begin
232                 if (nXreq = 'ndreq ') then
233                 begin
234                   stateXa := one;              (* back to idle *)
235                   wsab(ptrwnd, wndno, 'd     ', id, true)        (* SEND data to NTRNSDB *)
236                 end
237                 else if (nXreq = 'ncres ') then
238                 begin
239                   stateXa := four;          (* dataXtransferXrdy *)
240                   wsab(ptrwnd, wndno, 's     ', id, true)    (* SEND data to NTRNSDB *)
241                 end
242                 else        (* erroneous net primitive sent by TPRENTA *)
243                             (* this situation will not arise *)
244                   wmess(6, stateXa, nXreq)
245               end
246               else    (* if NOT queue then write for debug purposes *)
247               begin
248                 writeln(tty, 'RECEIVE by ref. does not work in INCONNECT PENDING');
249                 writeln(tty, 'state; only strings such as NCRES etc are accepted')
250               end
251             end;                  (* INXCONNXPENDING *)
252
253
254         (* if in dataXtransferXrdy state, then do the following *)
255
256
257      four : begin
258               if (queue) then                   (* RECEIVE data *)
259               begin
260                 if (nXreq = 'ndreq ') then
261                 begin
```

```
262                 wqab(ptrwnd, wndno, 'd     ', id, true);      (* SEND data to NTRNSDB *)
263                 stateXa := one                (* back to idle *)
264               end
265               else                          (* erroneous net primitive received *)
266                                             (* from TPRENTA *)
267                                             (* this situation will not arise *)
268               wmess(6, stateXa, nXreq)
269             end
270             else                              (* RECEIVE by ref. *)
271             begin
272                                               (* concatenate a single
273                                                  TPDU in USER DATA field
274                                                  of a NET SERV. PRIMITIVES *)
275               wqab(ptrwnd, wndno, 'nmodr ', id, false)          (* SEND by ref. *)
276
277             end                               (* RECEIVE by ref. *)
278           end                                 (* dataXtransferXrdy *)
279         end                                   (* case *)
280       end                                     (* incoming  user message *)
281     (*
282
283
284       the following section deals with the messages coming in from the
285       queueXbXtoXa
286
287     *)
288       else if (nXreq[1] = 'e') then
289         wmess(7, stateXa, nXreq)
290       else case stateXa of
291
292     (* if in idle state, then do the following *)
293
294       one:begin
295             if (queue) then             (* RECEIVE data *)
296             begin
297               if (nXreq[1] = 'a') then
298               begin
299                 sdtprenta(ptrwnd, wndno, 'ncind ', id, true);  (* SEND data NCIND to TPRENTA *)
300                 stateXa := three                  (* inXconnXpending *)
301               end
302               else                  (* anything other than a 'a' *)
303               begin
304                 wqab(ptrwnd, wndno, 'e     ', id, true)           (* SEND data to NTRNSDB *)
```

```
305             end
306           end
307           else     (* if NOT queue then write for debug purposes *)
308           begin
309             writeln(tty, 'RECEIVE by ref. does not work in IDLE');
310             writeln(tty, 'state; only strings such as NCRES etc are accepted')
311           end
312         end;                                (* IDLE *)
313
314
315       (* if in outXconnXpending state, then do the following     *)
316
317
318     two:begin
319           if (queue) then
320           begin
321             if (nXreq[1] = 'd') then
322             begin
323               wmess(3, stateXs, nXreq);
324               stateXs := one                  (* back to idle *)
325             end
326             else if (nXreq[1] = 's') then
327             begin
328               sdtprenta(ptrwnd, wndno, 'nccon ', id, true);  (* SEND data NCCON to TPRENTA *)
329               stateXs := four               (* dataXtransferXrdy *)
330             end
331             else
332             begin
333               wasb(ptrwnd, wndno, 'e     ', id, true)        (* SEND data to NTRNSDB *)
334             end
335           end                   (* queue *)
336           else     (* if NOT queue then write for debug purposes *)
337           begin
338             writeln(tty, 'RECEIVE by ref. does not work in OUTCONNECT PENDING');
339             writeln(tty, 'state; only letters such as "s" or "d" are accepted')
340           end
341         end;                                  (* OUTXCONNXPENDING *)
342
343
344       (*      if in inXconnXpending state, then do the following    *)
345
346     three  :
347         begin
```

```
348             if (queue) then       (* RECEIVE data from NTRNSDB *)
349             begin
350               if (nXreq[1] = 'd') then
351               begin
352                 sdtprenta(ptrwnd, wndno, 'ndind ', id, true);       (* SEND data NDIND to TPRENTA *)
353                 stateXa := one
354               end
355               else
356               begin
357                 waab(ptrwnd, wndno, 'e     ', id, true)      (* SEND data to NTRNSDB   *)
358               end
359             end
360             else     (* if NOT queue then write for debug purposes *)
361             begin
362               writeln(tty, 'RECEIVE by ref. does not work in INCONNECT PENDING');
363               writeln(tty, 'state; only letters such as a "d" are accepted')
364             end
365           end;                                  (* inXconnXpending *)
366
367
368         (*      if in dataXtransferXrdy state, then do the following   *)
369
370   four:   begin
371             if (queue) then
372             begin
373               if (nXreq[1] = 'd') then
374               begin
375                 sdtprenta(ptrwnd, wndno, 'ndind ', id, true);(* SEND data NDIND to TPRENTA *)
376                 stateXa := one                    (* back to idle *)
377               end
378               else if (nXreq[1] = 'e') then
379               begin
380                 waab(ptrwnd, wndno, 'e     ', id, true) (* SEND data to NTRNSDB *)
381               end
382             end
383             else                (* RECEIVE by ref. *)
384             begin
385               if (nXreq = 'nnodr ') then
386                 sdtprenta(ptrwnd, wndno, 'nnodr ', id, false)
387                                   (* SEND by ref. to tprenta.
388                                     Disintegration of the
389                                     net primitive i.e NNODR with
390                                     a TPDU in its user data field
```

```
  391                                        separation
  392                                        is not explicitly done but
  393                                        the relevant portion of the
  394                                        received net primitive
  395                                        i.e the TPDU is read by the
  396                                        tprenta          *)
  397
  398          else                       (* erroneous net primitive sent
  399                                        by NTRNSDB , This situation will
  400                                        not arise *)
  401            begin
  402              sdtprenta(ptrwnd, wndno, 'ndind '; id, true);  (* SEND data NDIND to TPRENTA *)
  403              stateXa := one
  404            end
  405        end                            (* RECEIVE by ref. *)
  406      end                              (* dataXtransferXrdy *)
  407    end                                (* case *)
  408
  409  end;                                 (* end of RUN forever *)
  410
  411  writeln(tty, 'NTRNSDA TASK TERMINATED.....................');
  412 end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      013702
OUTERMOST DATA SIZE     000124
RESERVED STACK & HEAP   001152
COMPILATION TIME 100 SECONDS

PAS NTRNSDA,NTRNSDA/P42=NTRNSDA
```

```
 1  (*$m-*)
 2  (*****************************************************************************
 3
 4      NAME:         PREPTPDU
 5
 6      FUNCTION:     To prepare a TPDU according to the TPDU code.
 7
 8      USAGE:        procedure preptpdu (ptrwnd : pstd;
 9                                        var dstref : arr;
10                                        var srcref : arr;
11                                        rjcse  : integer;
12                                        err : errr;
13                                        cod : integer);
14                            .
15                            .
16                            .
17                    preptpdu(ptrwnd, dstref, srcref, rjcse, err, cod);
18
19      INPUT PARAMETERS:     ptrwnd  - the pointer to the packet to be filled.
20                            dstref  - destination address.
21                            srcref  - source address.
22                            rjcse   - cause for rejection of TPDU.
23                            err     - length of the erroneous TPDU string.
24                            cod     - the type of TPDU packet.
25
26      OUTPUT PARAMETERS:    ptrwnd  - the pointer to the TPDU.
27                            dstref  - destination address.
28                            srcref  - source address.
29
30      MODULES CALLED:       (The symbol '>' designates a primitive)
31
32                            >chr
33                            >writeln
34
35      EXIT CONDITIONS:
36                            Normal exit conditions.
37
38      REMARKS:
39                    Presently, most of the filling is done in TERMA.  For futur
40                    enhancements, the filling should be done by this module.
41
42                    See the TPRENTA module for more information about the format
43                    of the different TPDUs.
```

```
   44
   45   ********************************************************************************)
   46
   47   {$I+TPRENTA.DEF}
```

```
    1
    2   const                           { GLOBAL CONSTANTS }
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           STRLEN     = 118;
    6           NETREQLEN  = 6;
    7           CRCODE     = 224;
    8           CCCODE     = 208;
    9           DTCODE     = 240;
   10           DRCODE     = 128;
   11           NDRCODE    = 120;
   12           ERCODE     = 112;
   13           APRNO      = 7;
   14           WNDSZ      = 3;
   15           REGSZ      = 21;  { NOPTR times WNDSZ }
   16           NOPTR      = 7;
   17           DUMLEN     = 10;
   18           DUMMLEN    = 5;
   19           NODATFLG   = 20;
   20           IVCODE     = 193;
   21           DVCODE     = 224;
   22           ERRLEN     = 128;
   23           GCDEST     = 2222;
   24           GCSOUR     = 1111;
   25           TPDUSZPL   = 1;
   26           TPDUSZPC   = 192;
   27           THPC       = 137;
   28           GCTPDU     = 128;
   29           GCTHR1     = 1;
   30           GCTHR2     = 2;
   31           GCTHR3     = 3;
   32           GCTHR4     = 4;
   33           THPL       = 12;
   34           TRDLPL     = 8;
   35           TRDLPC     = 136;
   36           GCTRD1     = 7;
   37           GCTRD2     = 8;
   38           GCTRD3     = 9;
   39           GCTRD4     = 10;
   40           NREUI      = 99;
   41           GCRESU     = 11;
   42           UNDEF      = 99;
```

```
44
45   type                          { GLOBAL TYPES }
46        std = record             { TPDUs }
47                fixp : packed array [1..FIXPLEN] of char;
48                oth  : packed array [1..OTHLEN] of char;
49        end;
50
51        pstd = ^std;
52
53        contrl = array[0..7] of integer; { format for CRRG$ }
54
55        netXreq   = packed array[1..NETREQLEN] of char;
56
57        ddd = record
58                req : netXreq;
59                dum : array[1..DUMMLEN] of integer;
60        end;
61
62        pdd = ^ddd;
63
64        tskk = record
65                nam : array [1..2] of integer;
66        end;
67
68        rbuff = record { format for RCVD$ }
69                nam : tskk;
70                req : netXreq;
71                dum : array [1..DUMLEN] of integer;
72        end;
73
74        buff = record  { format for SDAT$ }
75                req : netXreq;
76                dum : array [1..DUMLEN] of integer;
77        end;
78
79        rddd = record
80                oke: array[1..2] of integer;
81                req : netXreq;
82                dum : array[1..DUMMLEN] of integer;
83        end;
84
85        rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rapr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rresid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  apr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  resid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmgg = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
130
131         errr = integer;
```

```
48
49
50  procedure preptpdu (ptrwnd : pstd;
51                      var dstref : arr;
52                      var srcref : arr;
53                      rjcse  : integer;
54                      err : errr;
55                      cod : integer);
56
57  var
58          i           : integer;
59          j           : integer;
60          temp        : std;
61
62  procedure fillw(b : integer;
63                  var c1 : char;
64                  var c2 : char); extern;
65
66  begin                                   (* PREPTPDU *)
67
68  { writeln(tty, 'entering.preptpdu routine');}
69
70    case cod of                           (* select the correct type of TPDU *)
71
72    CRCODE :                              (* prepare CR-TPDU *)
73      begin                               (* CRCODE *)
74        dstref[1] := ptrwnd^.fixp[3];     (* save the source and   *)
75        dstref[2] := ptrwnd^.fixp[4];     (* destination addresses *)
76        srcref[1] := ptrwnd^.fixp[5];     (* for futur references  *)
77        srcref[2] := ptrwnd^.fixp[6];
78        ptrwnd^.fixp[2] := chr(42)        (*  fill up LI field   *)
79      end;                                (* CRCODE *)
80
81    CCCODE :                              (* prepare CC-TPDU *)
82      begin                               (* CCCODE *)
83        ptrwnd^.fixp[2] := chr(42);       (*  fill up LI field   *)
84        ptrwnd^.fixp[3] := dstref[1];     (* put in the proper destination *)
85        ptrwnd^.fixp[4] := dstref[2]
86      end;                                (* CCCODE *)
87
88    DTCODE :                              (* prepare DT-CODE *)
89      begin                               (* DTCODE *)
90        ptrwnd^.fixp[2] := chr(2)         (*  fill up LI field   *)
91      end;                                (* DTCODE *)
```

```
 92
 93     ERCODE :                                (* prepare ER-TPDU *)
 94       begin                                 (* ERCODE *)
 95         ptrwnd^.fixp[1] := chr(ERCODE);
 96         ptrwnd^.fixp[2] := chr(130);        (*     LI     *)
 97         ptrwnd^.fixp[3] := dstref[1];       (* fill up the fixed part of *)
 98         ptrwnd^.fixp[4] := dstref[2];       (* the ER TPDU               *)
 99         ptrwnd^.fixp[6] := chr(rjcse);      (* fill the rejection cause field *)
100         ptrwnd^.oth[2]  := chr(IVCODE);
101         (* fill up the variable part of the ER TPDU up to where the error was
102            detected *)
103         (* assuming OTHLEN - 2 = ERRLEN *)
104         temp.fixp := ptrwnd^.fixp;
105         temp.oth := ptrwnd^.oth;
106
107         if err >= FIXPLEN then
108         begin
109
110           for i := 1 to FIXPLEN do
111             ptrwnd^.oth[i + 2] := temp.fixp[i];
112           for i := (FIXPLEN + 1) to err do
113             ptrwnd^.oth[i + 2] := temp.oth[i];
114           for i := (err + 1) to ERRLEN do
115             ptrwnd^.oth[i + 2] := chr(UNDEF)
116         end
117         else
118         begin
119           for i := 1 to err do
120             ptrwnd^.oth[i + 2] := temp.fixp[i];
121           for i := (err + 1) to FIXPLEN do
122             ptrwnd^.oth[i + 2] := chr(UNDEF);
123           for i := (FIXPLEN + 1) to ERRLEN do
124             ptrwnd^.oth[i + 2] := chr(UNDEF)
125         end
126       end;                                  (* ERCODE *)
127
128     DRCODE :                                (* prepare DR-TPDU *)
129       begin                                 (* DRCODE *)
130         ptrwnd^.fixp[2] := chr(127)         (*   LI       *)
131       end;                                  (* DRCODE *)
132
133     NDRCODE :
134       begin                                 (* prepare complete DR-TPDU *)
```

```
 135          ptrwnd^.fixp[1] := chr(DRCODE);
 136          ptrwnd^.fixp[2] := chr(127);      (*   LI       *)
 137          fillw(GCDEST, ptrwnd^.fixp[3], ptrwnd^.fixp[4]);
 138          fillw(GCSOUR, ptrwnd^.fixp[5], ptrwnd^.fixp[6]);
 139          ptrwnd^.fixp[8] := chr(GCRESO);
 140          ptrwnd^.oth[1] := chr(1);         (* DVL *)
 141          ptrwnd^.oth[2] := chr(DVCODE);
 142          ptrwnd^.oth[3] := chr(7)           )
 143        end
 144
 145      end                                   (* case *)
 146
 147    { writeln(tty, 'leaving preptpdu routine');}
 148    end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      004424
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000740
COMPILATION TIME 51 SECONDS

PAS PREPTPDU,PREPTPDU/P42=PREPTPDU
```

```
    1   (*$n-*)
    2   (*****************************************************************************
    3
    4      NAME:         READuDATA
    5
    6      FUNCTION:     To read in a string from standard input into the appropriate
    7                    field of a particular packet, record its length and put it into
    8                    the correct length field of this packet.
    9
   10      USAGE:        procedure  readUdata(var w : std);extern;
   11                                .
   12                                .
   13                                .
   14                    readUdata(w);
   15
   16      INPUT PARAMETERS:    NONE
   17
   18      OUTPUT PARAMETERS:   w       - the packet to be filled with a string and its
   19                                     associated length.
   20
   21      MODULES CALLED:
   22                           >break
   23                           >eoln
   24                           >read
   25                           >readln
   26                           >writeln
   27                           fillb
   28
   29      EXIT CONDITIONS:     Normal condition.
   30
   31      REMARKS:
   32                    The inputed line is truncated to STRLEN, so care should
   33                    be taken not to exceed this value.
   34
   35                    The user (TERMINAL A) should make sure that his/her terminal
   36                    buffer is at least >= STRLEN.  This can be checked by issuing
   37                    the MCR command 'SET /BUF-TI:'.  If the buffer is too small,
   38                    then this can be altered by the MCR command 'SET /BUF-TI:200.'
   39                    if a buffer of 200 bytes is desired for example.
   40
   41                    Unpredictable behavior results when non-printable characters
   42                    , such as escape or end-of-file characters, are inputed by
   43                    this module.
```

```
44
45  ********************************************************************************)
46
47  const                           { GLOBAL CONSTANTS }
48          STRLEN  = 118;
49          FIXPLEN = 8;
50          OTHLEN  = 130;
51  type                            { GLOBAL TYPES }
52          std = record
53                  fixp : packed array [1..FIXPLEN] of char;
54                  oth : packed array [1..OTHLEN] of char;
55          end;
56
57  procedure fillb(b : integer;
58                  var c1 : char); extern;
59
60  procedure  readUdata(var w : std);
61
62  const                           { LOCAL VT100 TERMINAL CONSTANTS }
63          revideo = '[7m';
64          video   = '[0m';
65          up      = '[A';
66
67  var     i   : integer;
68          len : integer;
69
70  begin
71          (* set input into reverse video *)
72          writeln(CHR(27), revideo, CHR(27), up);
73
74          (* read the input line and record its length *)
75          break(tty);
76          readln(tty);
77          i := 0;
78          if not(eoln(ttyin)) then
79          begin
80                  i :=  1;
81                  (* length of string must not exceed STRLEN *)
82                  while (not (eoln(ttyin)) and (i <= STRLEN)) do
83                  begin
84                          read(tty, w.oth[2+i]);
85                          i := i + 1
86                  end;
```

```
   87                   len := i - 1;
   88           end
   89           else    (* empty line *)
   90                   len := i;
   91
   92           (* put the length in the appropriate fields *)
   93           fillb(len, w.oth[1]);
   94
   95           (* put back in normal display mode *)
   96           writeln(CHR(27), video, CHR(27), up)
   97   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      000712
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000524
COMPILATION TIME 12 SECONDS

PAS READAT,READAT/P42=READAT
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:        READTERM
    5
    6      FUNCTION:    To read in a string from standard input and record its length.
    7
    8      USAGE:       procedure  request(var buffer : request ; var len : integer);extern;
    9                                .
   10                                .
   11                                .
   12                   readterm(buffer, len);
   13
   14      INPUT PARAMETERS:    NONE
   15
   16      OUTPUT PARAMETERS:   buffer          - the string buffer.
   17                           len             - the length of the string buffer.
   18
   19      MODULES CALLED:
   20                           >break
   21                           >eoln
   22                           >read
   23                           >readln
   24                           >writeln
   25
   26      EXIT CONDITIONS:     Normal condition.
   27
   28      REMARKS:
   29                   The inputed line is truncated to REQLEN, so care should
   30                   be taken not to exceed this value.
   31
   32                   It is not recommended to input non-printable characters using
   33                   this module since unpredictable behavior can result when these
   34                   strings are outputed.
   35
   36  *****************************************************************************)
   37
   38  const                                          { GLOBAL CONSTANT }
   39          REQLEN = 6;
   40  type                                           { GLOBAL TYPE }
   41          request = array [1..REQLEN] of char;
   42
   43  procedure  readterm(var buffer : request ; var len : integer);
```

```
   44
   45   const                                    { LOCAL VT100 TERMINAL CONSTANTS }
   46           revideo = '[7m';
   47           video   = '[0m';
   48           up      = '[A';
   49
   50   var     i : integer;
   51
   52   begin
   53           (* set input into reverse video *)
   54           writeln(CHR(27), revideo, CHR(27), up);
   55
   56           (* read the input line and record its length *)
   57           break(tty);
   58           readln(tty);
   59           i := 0;
   60           if not(eoln(ttyin)) then
   61           begin
   62                   i :=  1;
   63                   (* length of string must not exceed REQLEN *)
   64                   while (not (eoln(ttyin)) and (i <= REQLEN)) do
   65                   begin
   66                           read(tty, buffer[i]);
   67                           i := i + 1
   68                   end;
   69                   len := i - 1;
   70           end
   71           else    (* empty line *)
   72                   len := i;
   73
   74           (* put back in normal display mode *)
   75           writeln(CHR(27), video, CHR(27), up)
   76   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      000640
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000522
COMPILATION TIME 11 SECONDS

PAS READTERM,READTERM/P42=READTERM
```

```
    1   (*$m-*)
    2   (*****************************************************************************)
    3
    4       NAME:         RECCA
    5
    6       FUNCTION:     To wait until some information is sent to the TPRENTA task
    7                     via send data or send by refefence, and receives it via
    8                     receive data or receive by reference.
    9
   10       USAGE:        procedure recca(var ptrwnd : psid;
   11                                    var nXreq : netXreq;
   12                                    var queue : boolean;
   13                                    var messXterma : boolean); extern;
   14                                .
   15                                .
   16                                .
   17                     recca(ptrwnd, nXreq, queue, messXterma);
   18
   19
   20       INPUT PARAMETERS      None
   21
   22       OUTPUT PARAMETERS:    ptrwnd  - pointer to received by reference buffer.
   23                              nXreq   - receive data or reference string.
   24                              queue   - true if a receive data was performed
   25                                        false if a receive by reference was done.
   26                              messXterma- true if data was sent by TERMA task
   27                                          false if data was sent by NTRNSDA task.
   28
   29       MODULES CALLED:       (The symbol '>' designates a primitive)
   30
   31                              >chr
   32                              >halt
   33                              >writeln
   34                              buffwait
   35                              craw
   36                              dtrg
   37                              mark
   38                              receiv
   39                              rref
   40
   41       EXIT CONDITIONS:
   42                              If there are any errors detected in calling CRAW$, DTRG$,
   43                              MARK$, RECEIV$ or RREF$, then an error message identifying
```

```
                        the error and error code is printed onto standard I/O and
                        the processor is then halted by using the 'halt' primitive.

                        The return code of all these system directives, is set to
                        greater or equal to 0(zero) in case of successful completion
                        and to less than 0 in case of unsuccessful completion.

                        For more information on the use of these system routines
                        and their return codes, see the RSX-11M V3.2 Executive
                        Reference Manual.

    REMARKS:      When there is a receive by reference, the newly attached region
                  is detached in order to prevent depletion of the memory pool.
                  See page 6-108 of Executive Reference Manual for more details.

*****************************************************************************)

{$I+TPRENTA.DEF}
```

```
    1
    2   const                          { GLOBAL CONSTANTS }
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           STRLEN     = 118;
    6           NETREQLEN  = 6;
    7           CRCODE     = 224;
    8           CCCODE     = 208;
    9           DTCODE     = 240;
   10           DRCODE     = 128;
   11           NDRCODE    = 120;
   12           ERCODE     = 112;
   13           APRNO      = 7;
   14           WNDSZ      = 3;
   15           REGSZ      = 21;   { NOPTR times WNDSZ }
   16           NOPTR      = 7;
   17           DUMLEN     = 10;
   18           DUMMLEN    = 5;
   19           NODATFLG   = 20;
   20           IVCODE     = 193;
   21           DVCODE     = 224;
   22           ERRLEN     = 128;
   23           GCDEST     = 2222;
   24           GCSOUR     = 1111;
   25           TPDUSZPL   = 1;
   26           TPDUSZPC   = 192;
   27           THPC       = 137;
   28           GCTPDU     = 128;
   29           GCTHR1     = 1;
   30           GCTHR2     = 2;
   31           GCTHR3     = 3;
   32           GCTHR4     = 4;
   33           THPL       = 12;
   34           TRDLPL     = 8;
   35           TRDLPC     = 136;
   36           GCTRD1     = 7;
   37           GCTRD2     = 8;
   38           GCTRD3     = 9;
   39           GCTRD4     = 10;
   40           NREOT      = 99;
   41           GCRESO     = 11;
   42           UNDEF      = 99;
   43
```

```
44
45   type                          { GLOBAL TYPES }
46        std = record             { TPDUs }
47                fixp : packed array [1..FIXPLEN] of char;
48                oth  : packed array [1..OTHLEN] of char;
49        end;
50
51        pstd = ^std;
52
53        contrl = array[0..7] of integer; { format for CRRG$ }
54
55        netXreq   = packed array[1..NETREQLEN] of char;
56
57        ddd = record
58                req : netXreq;
59                dum : array[1..DUMMLEN] of integer;
60        end;
61
62        pdd = ^ddd;
63
64        tskk = record
65                nam : array [1..2] of integer;
66        end;
67
68        rbuff = record { format for RCVD$ }
69                nam : tskk;
70                req : netXreq;
71                dum : array [1..DUMLEN] of integer;
72        end;
73
74        buff = record  { format for SDAT$ }
75                req : netXreq;
76                dum : array [1..DUMLEN] of integer;
77        end;
78
79        rddd = record
80                ok : array[1..2] of integer;
81                req : netXreq;
82                dum : array[1..DUMMLEN] of integer;
83        end;
84
85        rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rresid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  resid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmss = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
  130
  131           errr = integer;
```

```
   62
   63
   64  procedure buffwait(efn : efnn); extern;
   65
   66  procedure recca(var ptrwnd : pstd;
   67                  var nXreq : netXreq;
   68                  var queue : boolean;
   69                  var messXterma : boolean);
   70
   71  var
   72          dummy   : efnn;
   73          efn     : efnn;
   74          gotit   : boolean;
   75          ids     : idss;
   76          irdb    : contrl;
   77          isrb    : rddd;
   78          rbuf    : rbuff;
   79          rwdb    : rwdbb;
   80          rrwdb   : rwdbb;
   81          tmg     : tmgg;
   82          tnt     : tntt;
   83          tsk     : tskk;
   84
   85
   86  procedure craw(var rwdb : rwdbb; var ids : idss); extern(fortran);
   87
   88  procedure dtrg(var irdb : contrl; var ids : idss); extern(fortran);
   89
   90  procedure mark(var efn : efnn;
   91                 var tmg : tmgg;
   92                 var tnt : tntt;
   93                 var ids : idss); extern(fortran);
   94
   95  procedure receiv(var tsk : tskk;
   96                   var rbuf : rbuff;
   97                   var dummy : efnn;
   98                   var ids : idss); extern(fortran);
   99
  100  procedure rref(var rwdb : rwdbb;
  101                 var isrb : rddd;
  102                 var ids : integer); extern(fortran);
  103
  104  begin
  105  { writeln(tty, ' entering recca procedure'); }
```

```
106
107     messXterma := false;    (* initialize booleans *)
108     queue := false;
109     gotit := false;
110     efn := NODATFLG;        (* initialize waiting local event flag *).
111     while not gotit do      (* do until something is received *)
112     begin
113
114       (* receive the data *)
115       receiv(tsk, rbuf, dummy, ids);
116   {   writeln(tty, 'receiv ids = ', ids); }
117       if ids <> -8 then
118       begin
119         if ids < 0 then
120         begin
121           writeln(tty, 'error in receiving data using "receiv" error code = ', ids);
122           halt
123         end
124         else
125         begin
126           nXrea := rbuf.rea;
127           gotit := true;
128           queue := true
129         end
130       end
131       else    (* receive by reference *)
132       begin
133         (* prepare to receive by reference *)
134
135         (* window definition block *)
136         rrwdb.rwstats := 0;         (* window status word *)
137                                     (* bits set: NONE     *)
138         rref(rrwdb, isrb, ids);
139 {       writeln(tty, 'rref ids = ', ids);}
140         if ids <> -8 then
141         begin
142           if ids < 0 then
143           begin
144             writeln(tty, ' error in using rref,  error code = ',ids);
145             halt
146           end
147           else
148           begin  (* successful receive by reference *)
```

```
149   {         writeln(tty, 'region id (pointer to attachment description=',ord(rrwdb.rregid));
150             writeln(tty, 'offset word specified by sender task=',rrwdb.roffs);
151             writeln(tty, 'length word specified by sender task=',rrwdb.rlen);
152             writeln(tty, 'window status word =', rrwdb.rwstats);
153             writeln(tty, 'virtual base address = ', ord(rrwdb.rvbadd));}
154
155             gotit := true;
156
157             (* create receive window into the newly attached region *)
158             rwdb.rapr[2] := chr(APRNO);   (* Active page register 7 *)
159             rwdb.rsiz := WNDSZ;           (* the size of the window in 32-word blocks *)
160             rwdb.rregid := rrwdb.rregid;  (* region id *)
161                                           (* or 0 for task region *)
162                                           (* specified only if WS.MAP is set *)
163             rwdb.roffs := rrwdb.roffs;    (* offset in region (32W blocks) at which
164                                              the window is to start mapping *)
165                                           (* specified only if WS.MAP is set *)
166             rwdb.rlen := rrwdb.rlen;      (* length to map (32W blocks) *)
167                                           (* or 0 if the length is to default to either
168                                      the size of the window or the space remaining
169                                      in the region *)
170                                           (* specified only if WS.MAP is set *)
171             rwdb.rwstats := 386;          (* window status word *)
172                                           (* bits set:
173                                              WS.64B = 256
174                                              WS.MAP = 128
175                                              WS.WRT = 2                  *)
176             crew(rwdb, ids);
177   {         writeln(tty, 'window created ids =', ids);
178             writeln(tty, 'Id assigned to the window = ',ord(rwdb.rapr[1]));
179             writeln(tty, 'virtual base address = ', ord(rwdb.rvbadd));
180             writeln(tty, 'length actually mapped in 32-word blocks = ', rwdb.rlen);
181             writeln(tty, 'window status word = ', rwdb.rwstats);}
182             if ids < 0 then
183             begin
184                writeln(tty, ' error in using crew,  error code = ',ids);
185                halt
186             end;
187
188             (* put receive packet in returned buffer *)
189             ptrwnd^.fixp := rwdb.rvbadd^.fixp;
190             ptrwnd^.oth := rwdb.rvbadd^.oth;
191
```

```
192             (* detach from newly attached region *)
193             irdb[0] := rwdb.rregid; (* region id *)
194             irdb[6] := 0;  (* no bits set *)
195             dtrg(irdb, ids);
196     {       writeln(tty, 'ids of detach is ', ids);
197             writeln(tty, 'detach status word =.', irdb[6]);}
198             if ids < 0 then
199             begin
200               writeln(tty, 'error trying to detach from newly attached region');
201               halt
202             end;
203
204             (* check if packet is from terma *)
205             if ((rrwdb.rsrbuff^.ok[1] = 32218) and (rrwdb.rsrbuff^.ok[2] = 20840)) then
206             begin
207     {         writeln(tty, 'message from terma');}
208               messXterma := true
209             end
210           end
211         end
212         else
213         begin           (* wait for a while and try again *)
214           tmg := 10; (* wait 10  *)
215           tnt := 2; (* seconds *)
216           mark(efn, tmg, tnt, ids);
217           buffwait(efn)
218         end
219       end
220     end
221
222   {  writeln(tty, ' leaving  recca procedure');}
223   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      002472
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000702
COMPILATION TIME 52 SECONDS

PAS`RECCA,RECCA/P42=RECCA
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         SDTPRENTA
    5
    6      FUNCTION:     To send information to task TPRENTA via system directives
    7                    send data (SDAT$) or send by reference (SREF$).
    8
    9      USAGE:        procedure sdtprenta(ptrwnd : pstd;
   10                                       var wndno : wndnoo;
   11                                       nXreq : netXreq;
   12                                       id : integer;
   13                                       bol : boolean); extern;
   14                                .
   15                                .
   16                                .
   17                   sdtprenta(ptrwnd, wndno, nXreq, id, bol);
   18
   19      INPUT PARAMETERS:   ptrwnd  - pointer to the buffer to be sent.
   20                          wndno   - the window number to be sent if bol = false.
   21                          nXreq   - the string to be sent.
   22                          id      - the region id.
   23                          bol     - the boolean switch.
   24
   25      OUTPUT PARAMETERS:  wndno   - the next window number.
   26
   27      MODULES CALLED:     (The symbol ">" designates a primitive)
   28
   29                          >halt
   30                          >writeln
   31                          craw
   32                          sref
   33                          sdat
   34
   35      EXIT CONDITIONS:
   36
   37                          If there are any errors detected in calling CRAW$, SDAT$
   38                          or SREF$, then an error message identifying the error and
   39                          error code is printed onto standard I/O and the processor
   40                          is then halted by using the 'halt' primitive.
   41
   42                          The return code of all these system directives, is set or
   43                          greater or equal to 0 (zero) in case of successful completion
```

```
                        and less than 0 in case of unsuccessful completion.

                        For more information on the use of these system routines
                        and their return codes, see the RSX-11M V3.2 Executive
                        Reference Manual.

    REMARKS:
                        See TSEND routine for information on method of circular
                        window queues and its restrictions.

*****************************************************************************)

{$I+TPRENTA.DEF}
```

```
 1
 2   const                          { GLOBAL CONSTANTS }
 3           FIXPLEN    = 8;
 4           OTHLEN     = 130;
 5           STRLEN     = 118;
 6           NETREQLEN  = 6;
 7           CRCODE     = 224;
 8           CCCODE     = 208;
 9           DTCODE     = 240;
10           DRCODE     = 128;
11           NDRCODE    = 120;
12           ERCODE     = 112;
13           APRNO      = 7;
14           WNDSZ      = 3;
15           REQSZ      = 21;  { NOPTR times WNDSZ }
16           NOPTR      = 7;
17           DUMLEN     = 10;
18           DUMMLEN    = 5;
19           NODATFLG   = 20;
20           IVCODE     = 193;
21           DVCODE     = 224;
22           ERRLEN     = 128;
23           GCDEST     = 2222;
24           GCSOUR     = 1111;
25           TPDUSZPL   = 1;
26           TPDUSZPC   = 192;
27           THPC       = 137;
28           GCTPDU     = 128;
29           GCTHR1     = 1;
30           GCTHR2     = 2;
31           GCTHR3     = 3;
32           GCTHR4     = 4;
33           THPL       = 12;
34           TRDLPL     = 8;
35           TRDLPC     = 136;
36           GCTRD1     = 7;
37           GCTRD2     = 8;
38           GCTRD3     = 9;
39           GCTRD4     = 10;
40           NREOT      = 99;
41           GCRESD     = 11;
42           UNDEF      = 99;
43
```

```
44
45   type                          { GLOBAL TYPES }
46          std = record           { TPDUs }
47                  fixp : packed array [1..FIXPLEN] of char;
48                  oth  : packed array [1..OTHLEN] of char;
49          end;
50
51          pstd = ^std;
52
53          contrl = array[0..7] of integer; { format for CRRG$ }
54
55          netXreq    = packed array[1..NETREQLEN] of char;
56
57          ddd = record
58                  req : netXreq;
59                  dum : array[1..DUMMLEN] of integer;
60          end;
61
62          pdd = ^ddd;
63
64          tskk = record
65                  nam : array [1..2] of integer;
66          end;
67
68          rbuff = record { format for RCVD$ }
69                  nam : tskk;
70                  req : netXreq;
71                  dum : array [1..DUMLEN] of integer;
72          end;
73
74          buff = record  { format for SDAT$ }
75                  req : netXreq;
76                  dum : array [1..DUMLEN] of integer;
77          end;
78
79          rddd = record
80                  ok : array[1..2] of integer;
81                  req : netXreq;
82                  dum : array[1..DUMMLEN] of ioteger;
83          end;
84
85          rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : rstd;
 90                  rsiz : integer;
 91                  rregid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : rstd;
101                  siz : integer;
102                  regid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : rdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          oprXsts     = (oprens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmsg = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
  130
  131           errr = integer;
```

```
 57
 58
 59   procedure buffwait(efn : efnn); extern;
 60
 61   procedure buffclear(efn : efnn); extern;
 62
 63   procedure buffso(efn : efnn); extern;
 64
 65   procedure sdtprenta(ptrwnd : pstd;
 66                       var wndno : wndnoo;
 67                       nXreq : netXreq;
 68                       id : integer;
 69                       bol : boolean);
 70
 71   var
 72           bf      : buff;
 73           efn     : efnn;
 74           ids     : idss;
 75           isrb    : ddd;
 76           swdb    : wdbb;
 77           tsk     : tskk;
 78           wdb     : wdbb;
 79
 80   procedure craw(var wdb : wdbb; var ids : idss); extern(fortran);
 81
 82   procedure send(var tsk : tskk;
 83                  var bf  : buff;
 84                  var efn : efnn;
 85                  var ids : idss); extern(fortran);
 86
 87   procedure sref(var tsk: tskk;
 88                  var efn : efnn;
 89                  var swdb : wdbb;
 90                  var isrb : ddd;
 91                  var ids : idss); extern(fortran);
 92   begin
 93
 94   { writeln(tty, ' entering sdtprenta procedure');}
 95
 96     (* set up task name = 'TPRENA' *)
 97     tsk.nam[1] := 32658;
 98     tsk.nam[2] := 8561;
 99
100     if bol then  (* send data *)
```

```
101     begin
102       (* put string in buffer to be sent *)
103       bf.req := nXreq;
104
105       (* send the data to the 'TPRENA' task *)
106       send(tsk, bf, efn, ids);
107   {   writeln(tty, 'send ids = ', ids);}
108       if ids < 0 then
109       begin
110          writeln(tty, 'error in calling SEND in SENDD,  error code = ', ids);
111          halt
112       end
113     end
114     else
115     begin (* send by reference to task 'TPRENA' *)
116
117       (* create a send window in the region *)
118       wdb.apr[2] := chr(APRNO);    (* Active page register 7 *)
119       wdb.siz := WNDSZ;            (* the size of the window in 32-word blocks *)
120       wdb.regid := id;             (* region id *)
121                                    (* or 0 for task region *)
122                                    (* specified only if WS.MAP is set *)
123       wdb.offs := (wndno-1)*WNDSZ;(* offset in region (32W blocks) at which
124                                      the window is to start mapping *)
125                                    (* specified only if WS.MAP is set *)
126       wdb.len := WNDSZ;            (* length to map (32W blocks) *)
127                                    (* or 0 if the length is to default to either
128                                      the size of the window or the space remaining
129                                      in the region *)
130                                    (* specified only if WS.MAP is set *)
131       wdb.wstats := 386;           (* window status word *)
132                                    (* bits set:
133                                              WS.64B = 256
134                                              WS.MAP = 128
135                                              WS.WRT = 2                *)
136       crew(wdb, ids);
137   {   writeln(tty, 'window created ids =', ids);
138       writeln(tty, 'Id assigned to the window = ',ord(wdb.apr[1]));
139       writeln(tty, 'virtual base address = ', ord(wdb.vbadd));
140       writeln(tty, 'length actually mapped in 32-word blocks = ', wdb.len);
141       writeln(tty, 'window status word = ', wdb.wstats);}
142       if ids < 0 then
143       begin
```

```
144          writeln(tty,' error in using craw,  error code = ',ids);
145          halt
146        end;
147
148        (* stick buffer into the send window *)
149        wdb.vbadd^.fixp := ptrwnd^.fixp;
150        wdb.vbadd^.oth := ptrwnd^.oth;
151
152        (* prepare window definition block for send by reference *)
153        swdb.regid := id;              (* id of the region to be sent by reference *)
154        swdb.offs := (wndno-1) * WNDSZ;(* offset in region (32W blocks) *)
155        swdb.len := WNDSZ;             (* length to map (32W blocks) *)
156        swdb;wstats := 3;              (* window status word *)
157                                       (* bits set:
158                                                  WS.WRT = 2
159                                                  WS.RED = 1          *)
160        sref(tsk, efn, swdb, isrb, ids);
161        if ids < 0 then
162        begin
163          writeln(tty, ' error in using sref,  error code = ',ids);
164          halt
165        end;
166
167        (* set up window number for next call *)
168        if wndno = NOPTR then
169          wndno := 1
170        else
171          wndno := wndno + 1;
172
173      end
174
175    { writeln(tty,' leaving sdtprenta procedure')}
176    end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      001704
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000642
COMPILATION TIME 28 SECONDS

PAS SDTPRENTA,SDTPRENTA/P42=SDTPRENTA
```

```
1   (*$m-*)
2   (*****************************************************************************
3
4      NAME:        SENDD
5
6      FUNCTION:    To send information to task NTRNSDA via system directives
7                   send data (SDAT$) or send-by reference (SREF$).
8
9      USAGE:       procedure sendd(ptrwnd : pstd;
10                                  var wndno : wndnoo;
11                                  nXreq : netXreq;
12                                  id : integer;
13                                  bol : boolean); extern;
14                              .
15                              .
16                              .
17                  sendd(ptrwnd, wndno, nXreq, id, bol);
18
19     INPUT PARAMETERS:    ptrwnd  - the pointer to the buffer to be sent.
20                          wndno   - the window number to be used.
21                          nXreq   - the string to be sent if bol = true.
22                          id      - the region id.
23                          bol     - the boolean switch.
24
25     OUTPUT PARAMETERS:   wndno   - the next window number.
26
27     MODULES CALLED:      (The symbol '>' designates a primitive)
28
29                          >halt
30                          >writeln
31                          craw
32                          sref
33                          sdat
34                  ;
35     EXIT CONDITIONS:
36                          If there are any errors detected in calling CRAW$, SDAT$
37                          or SREF$, then an error message identifying the error and
38                          error code is printed onto standard I/O and the processor
39                          is then halted by using the 'halt' primitive.
40
41                          The return code of all these system directives, is set or
42                          greater or equal to 0 (zero) in case of successful completion
43                          and less than 0 in case of unsuccessful completion.
```

```
   44
   45                          For more information on the use of these system routines
   46                          and their return codes, see the RSX-11M V3.2 Executive
   47                          Reference Manual.
   48
   49      REMARKS:      NONE
   50
   51   ********************************************************************************)
   52
   53   {$I+TPRENTA.DEF}
```

```
    1
    2   const                          { GLOBAL CONSTANTS }
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           STRLEN     = 118;
    6           NETREQLEN  = 6;
    7           CRCODE     = 224;
    8           CCCODE     = 208;
    9           DTCODE     = 240;
   10           DRCODE     = 128;
   11           NDRCODE    = 120;
   12           ERCODE     = 112;
   13           APRNO      = 7;
   14           WNDSZ      = 3;
   15           REGSZ      = 21;  { NOPTR times WNDSZ }
   16           NOPTR      = 7;
   17           DUMLEN     = 10;
   18           DUMMLEN    = 5;
   19           NODATFLG   = 20;
   20           IVCODE     = 193;
   21           DVCODE     = 224;
   22           ERRLEN     = 128;
   23           GCDEST     = 2222;
   24           GCSOUR     = 1111;
   25           TPDUSZPL   = 1;
   26           TPDUSZPC   = 192;
   27           THPC       = 137;
   28           GCTPDU     = 128;
   29           GCTHR1     = 1;
   30           GCTHR2     = 2;
   31           GCTHR3     = 3;
   32           GCTHR4     = 4;
   33           THPL       = 12;
   34           TRDLPL     = 8;
   35           TRDLPC     = 136;
   36           GCTRD1     = 7;
   37           GCTRD2     = 8;
   38           GCTRD3     = 9;
   39           GCTRD4     = 10;
   40           NREOT      = 99;
   41           GCRESO     = 11;
   42           UNDEF      = 99;
   43
```

```
44
45   type                            { GLOBAL TYPES }
46        std = record               { TPDUs }
47                fixp : packed array [1..FIXPLEN] of char;
48                oth  : packed array [1..OTHLEN] of char;
49        end;
50
51        pstd = ^std;
52
53        contrl = array[0..7] of integer; { format for CRRG$ }
54
55        netXreq  = packed array[1..NETREQLEN] of char;
56
57        ddd = record
58                req : netXreq;
59                dum : array[1..DUMMLEN] of integer;
60        end;
61
62        pdd = ^ddd;
63
64        tskk = record
65                nam : array [1..2] of integer;
66        end;
67
68        rbuff = record { format for RCVD$ }
69                nam : tskk;
70                req : netXreq;
71                dum : array [1..DUMLEN] of integer;
72        end;
73
74        buff = record  { format for SDAT$ }
75                req : netXreq;
76                dum : array [1..DUMLEN] of integer;
77        end;
78
79        rddd = record
80                ok : array[1..2] of integer;
81                req : netXreq;
82                dum : array[1..DUMMLEN] of integer;
83        end;
84
85        rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rresid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : pdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  resid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmss = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
  130
  131           errr = inteserf
```

```
54
55
56   procedure sendd(ptrwnd : pstd;
57                   var wndno : wndnoo;
58                   nXreq : netXreq;
59                   id : integer;
60                   bol : boolean);
61
62   var
63           bf      : buff;
64           efn     : efnn;
65           ids     : idss;
66           isrb    : ddd;
67           swdb    : wdbb;
68           tsk     : tskk;
69           wdb     : wdbb;
70
71   procedure craw(var wdb : wdbb; var ids : idss); extern (fortran);
72
73   procedure send(var tsk : tskk;
74                  var bf  : buff;
75                  var efn : efnn;
76                  var ids : idss); extern(fortran);
77
78   procedure sref(var tsk: tskk;
79                  var efn : efnn;
80                  var swdb : wdbb;
81                  var isrb : ddd;
82                  var ids : idss); extern(fortran);
83   begin
84
85   { writeln(tty, ' entering sendd procedure');}
86
87     (* set up task name = 'TRNSDA' *)
88     tsk.nam[1] := 32734;
89     tsk.nam[2] := 30561;
90
91     if bol then  (* send data *)
92     begin
93       (* put string in buffer to be sent *)
94       bf.req := nXreq;
95
96       (* send the data to the 'TRNSDA' task *)
97       send(tsk, bf, efn, ids);
```

```
   98       if ids < 0 then
   99       begin
  100          writeln(tty, 'error in calling SEND in SENDD,  error code = ', ids);
  101          halt
  102       end
  103     end
  104     else
  105     begin
  106       (* create a send window in the region *)
  107       wdb.apr[2] := chr(APRNO);    (* Active page register 7 *)
  108       wdb.siz := WNDSZ;            (* the size of the window in 32-word blocks *)
  109       wdb.regid := id;             (* region id *)
  110                                    (* or 0 for task region *)
  111                                    (* specified only if WS.MAP is set *)
  112       wdb.offs := (wndno-1)*WNDSZ;(* offset in region (32W blocks) at which
  113                                      the window is to start mapping *)
  114                                    (* specified only if WS.MAP is set *)
  115       wdb.len := WNDSZ;            (* length to map (32W blocks) *)
  116                                    (* or 0 if the length is to default to either
  117                                      the size of the window or the space remaining
  118                                      in the region *)
  119                                    (* specified only if WS.MAP is set *)
  120       wdb.wstats := 386;           (* window status word *)
  121                                    (* bits set:
  122                                               WS.64B = 256
  123                                               WS.MAP = 128
  124                                               WS.WRT = 2                *)
  125       crew(wdb, ids);
  126   {   writeln(tty, 'window created ids =', ids);
  127       writeln(tty, 'Id assigned to the window = ',ord(wdb.apr[1]));
  128       writeln(tty, 'virtual base address = ', ord(wdb.vbadd));
  129       writeln(tty, 'length actually mapped in 32-word blocks = ', wdb.len);
  130       writeln(tty, 'window status word = ', wdb.wstats);}
  131       if ids < 0 then
  132       begin
  133          writeln(tty, ' error in using crew,  error code = ',ids);
  134          halt
  135       end;
  136
  137       (* stick buffer into the send window *)
  138       wdb.vbadd^.fixp := ptrwnd^.fixp;
  139       wdb.vbadd^.oth := ptrwnd^.oth;
  140
```

```
  141       (* prepare window definition block to send by reference *)
  142       swdb.regid := id;           (* id of the region to be sent by reference *)
  143       swdb.offs := (wndno-1)*WNDSZ;(* offset in region (32W blocks) *)
  144       swdb.len := WNDSZ;          (* length to map (32W blocks) *)
  145       swdb.wstats := 3;           (* window status word *)
  146                                   (* bits set:
  147                                              WS.WRT = 2
  148                                              WS.RED = 1        *)
  149       sref(tsk, efn, swdb, isrb, ids);
  150   {   writeln(tty, 'ids for sref is - ', ids);}
  151       if ids < 0 then
  152       begin
  153          writeln(tty, ' error in using sref, error code = ',ids);
  154          halt
  155       end;
  156
  157       (* set up window number to point to the next window in the circular queue *)
  158       if wndno = NOPTR then
  159         wndno := 1
  160       else
  161         wndno := wndno + 1;
  162
  163     end
  164
  165   { writeln(tty,' leaving sendd procedure')}
  166   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      001704
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000642
COMPILATION TIME 28 SECONDS

PAS SENDD,SENDD/P42=SENDD
```

```
    1   (*****************************************************************************
    2
    3      NAME:         TERMA (MAIN)
    4
    5      FUNCTION:     This module is used to continuously READ information from
    6                    USER A (TERMINAL A) and to send this information to the TPRENTA
    7                    task.  This information is obtained from terminal input, in the
    8                    form of user primitive strings and also data strings (see tnodr
    9                    and tdreq primitives), and also from certain text files which
   10                    contain information related to the selected user primitive.
   11
   12      USAGE:        from MCR : 'RUN TERMA /TASK=TERMA /INC=100.'
   13
   14      INPUT PARAMETERS:    NONE
   15
   16      OUTPUT PARAMETERS:   NONE
   17
   18      MODULES CALLED:      (The symbol '>' designates a primitive)
   19
   20                           >chr
   21                           >halt
   22                           >ioresult
   23                           >read
   24                           >readln
   25                           >reset
   26                           >writeln
   27                           buffclear
   28                           buffgo
   29                           buffwait
   30                           fillb
   31                           fillw
   32                           initbuf
   33                           readterm
   34                           readUdata
   35                           tinit
   36                           tsend
   37
   38      EXIT CONDITIONS:
   39                           This program has been designed to run indefinitely until
   40                           the task into which it runs is specifically aborted with
   41                           the MCR command 'ABORT TERMA'.
   42
   43                           Whenever there is an error detected, an error message
```

```
   44                        identifying the error and error code is printed onto
   45                        standard I/O and the processor is then halted by using
   46                        the 'halt' primitive.  Here is a list of the possible
   47                        errors :
   48
   49                        - file 'TCREQ.TXT' could not be reset properly.
   50                        - file 'TCRES.TXT' could not be reset properly.
   51                        - file 'TDREQ.TXT' could not be reset properly.
   52
   53                        If any errors are detected by one of the routines called,
   54                        then the routine itself will print an error message and
   55                        halt the processor.
   56
   57      REMARKS:
   58                        The files 'TCREQ.TXT', 'TCRES.TXT' and 'TDREQ.TXT' are
   59                        formatted in accordance with the specifications of this
   60                        module. Any changes in the format of this module or
   61                        of those text files should be done very carefully.
   62
   63    ********************************************************************************)
   64
   65    program terma(tty, input);
   66
   67    {$I+TERMA.DEF}
```

```
 1
 2  const
 3          REQLEN    = 6;
 4          NOPTR     = 7;
 5          WNDSZ     = 3;
 6          APRNO     = 7;
 7          REGSZ     = 21;
 8          FIXPLEN   = 8;
 9          OTHLEN    = 130;
10          STRLEN    = 118;
11          CRCODE    = 224;
12          CCCODE    = 208;
13          DTCODE    = 240;
14          DRCODE    = 128;
15          DVCODE    = 224;
16          CLGTSAPPC = 193;
17          CLGTSAPPL = 1;
18          CLDTSAPPC = 194;
19          CLDTSAPPL = 1;
20          TPDUSZPC  = 192;
21          TPDUSZPL  = 1;
22          THPC      = 137;
23          THPL      = 12;
24          TRDLPC    = 136;
25          TRDLPL    = 8;
26          NREOT     = 99;
27
28  type
29          std = record
30                  fixp : packed array [1..FIXPLEN] of char;
31                  oth : packed array [1..OTHLEN] of char;
32          end;
33
34          pstd = ^std;
35
36          contrl = array[0..7] of integer;
37
38          ddd = record
39                  ok : array[0..7] of integer;
40          end;
41
42          pdd = ^ddd;
43
```

```
 44          wdbb = record  { window block format }
 45                  arr : packed array[1..2] of char;
 46                  vbadd : pstd;
 47                  siz : integer;
 48                  regid : integer;
 49                  offs : integer;
 50                  len : integer;
 51                  wstats : integer;
 52                  srbuff : pdd;
 53          end;
 54
 55          tskk = record
 56                  nam : array[1..2] of integer;
 57          end;
 58
 59          efnn = integer;
 60
 61          idss = integer;
 62
 63          request = packed array[1..REQLEN] of char;
 64
 65          wndnoo = 1..NOPTR;
 66
```

```
 68
 69
 70  var
 71          a          : integer;
 72          id         : integer;
 73          j          : integer;
 74          len        : integer;
 75          rwndno     : std;
 76          swndno     : std;
 77          userXprim  : request;
 78          valid      : boolean;
 79          wndno      : wndnoo;
 80
 81
 82  procedure buffclear(efn : efnn); extern;
 83
 84  procedure buffgo(efn : efnn); extern;
 85
 86  procedure buffwait(efn : efnn); extern;
 87
 88  procedure fillb(b : integer;
 89                  var c : char);extern;
 90
 91  procedure fillw(b : integer;
 92                  var c1 : char;
 93                  var c2 : char);extern;
 94
 95  procedure initbuf(var swndno : std); extern;
 96
 97  procedure readterm(var userXreq : request;
 98                     var len : integer); extern;
 99
100  procedure readUdata(var wp : std); extern;
101
102  procedure tinit(var id : integer); extern;
103
104  procedure tsend(swndno : std;
105                  var wndno : wndnoo;
106                  id : integer); extern;
107
108  begin                                          (* TERMA *)
109    (* for display purposes on VT100 compatible terminals *)
110  { writeln(tty, chr(27),'[2J',
111                 chr(27), '[0;0f', chr(27), '[0m',
```

```
112                 chr(27), '[7;24r', chr(27), '[7m', chr(27), '[A');
113     writeln(tty, '*****************************************************************************************');
114     writeln(tty, '*                                                                                       *');
115     writeln(tty, '*                        T A S K   T E R M A  activated                                 *');
116     writeln(tty, '*                                                                                       *');
117     writeln(tty, '*****************************************************************************************');
118     writeln(tty, chr(27),'[0m');
119
120     wndno := 1;          (* initialize this to anything in valid range *)
121     tinit(id);           (* create a dynamic region with name 'A1' *)
122
123     while (1 = 1) do     (* DO forever *)
124     begin
125
126       (* initialize send buffer *)
127       initbuf(swindno);
128
129       valid := true;
130       (* read type of user primitive *)
131       writeln(tty,'readin users primitive type');
132       readterm(userXprim, len);
133
134       (* pad with blanks if needed *)
135       for j := (len + 1) to REQLEN do
136         userXprim[j] := ' ';
137
138       if userXprim = 'tcreq  ' then                                    (* tcreq *)
139       begin
140         (* go in the tcreq file for input *)
141         reset(input, 'TCREQ.TXT');
142         if ioresult(input) <> 0 then
143         begin
144           writeln(tty, 'Error opening the file called TCREQ.TXT ');
145           halt
146         end;
147
148         (* fill up the buffer with the appropriate fields *)
149         fillb(CRCODE, swindno.fixp[1]);
150   {     writeln(tty,'enter the DESTINATION address');}
151         readln(input);  (* skip over comment *)
152         readln(input, a);
153         fillw(a, swindno.fixp[3], swindno.fixp[4]);
154   {     writeln(tty,'enter the SOURCE address');}
```

```
  155          readln(input);  (* skip over comment *)
  156          readln(input, a);
  157          fillw(a, swndno.fixp[5], swndno.fixp[6]);
  158    {     writeln(tty,'enter the CLASS and OPTION');}
  159          readln(input);  (* skip over comment *)
  160          readln(input, a);
  161          fillb(a, swndno.fixp[8]);
  162    {     writeln(tty,'enter the CALLING TSAP identifier');}
  163          readln(input);  (* skip over comment *)
  164          readln(input, a);
  165          fillb(a, swndno.oth[4]);
  166          fillb(CLGTSAPPC, swndno.oth[2]);
  167          fillb(CLGTSAPPL, swndno.oth[1]);
  168    {     writeln(tty,'enter the CALLED TSAP identifier');}
  169          readln(input);  (* skip over comment *)
  170          readln(input, a);
  171          fillb(a, swndno.oth[5]);
  172          fillb(CLDTSAPPC, swndno.oth[3]);
  173          fillb(CLDTSAPPL, swndno.oth[6]);
  174    {     writeln(tty,'enter the TPDU size');}
  175          readln(input);  (* skip over comment *)
  176          readln(input, a);
  177          fillb(a, swndno.oth[10]);
  178          fillb(TPDUSZPL, swndno.oth[7]);
  179          fillb(TPDUSZPC, swndno.oth[8]);
  180
  181    {     writeln(tty,'enter the 4 THROUGHPUT values (8 octets)');}
  182          readln(input);  (* skip over comment *)
  183          read(input, a);
  184          fillw(a, swndno.oth[11], swndno.oth[12]);
  185          read(input, a);
  186          fillw(a, swndno.oth[13], swndno.oth[14]);
  187          read(input, a);
  188          fillw(a, swndno.oth[15], swndno.oth[16]);
  189          readln(input, a);
  190          fillw(a, swndno.oth[17], swndno.oth[18]);
  191
  192          fillb(THPC, swndno.oth[9]);
  193          fillb(THPL, swndno.oth[23]);
  194
  195    {     writeln(tty,'enter the 4 TRANSIT DELAY values (8 octets)');}
  196          readln(input);  (* skip over comment *)
  197          read(input, a);
```

```
198         fillw(a, swndno.oth[27], swndno.oth[28]);
199         read(input, a);
200         fillw(a, swndno.oth[29], swndno.oth[30]);
201         read(input, a);
202         fillw(a, swndno.oth[31], swndno.oth[32]);
203         readln(input, a);
204         fillw(a, swndno.oth[33], swndno.oth[34]);
205
206         fillb(TRDLPC, swndno.oth[25]);
207         fillb(TRDLPL, swndno.oth[24]);
208       end                                                    (* tcreu *)
209       else if userXprim = 'tcres ' then                      (* tcres *)
210       begin
211         (* go in the tcres file for input *)
212         reset(input, 'TCRES.TXT');
213         if ioresult(input) < 0 then
214         begin
215           writeln(tty, 'Error opening the file called TCRES.TXT ');
216           halt
217         end;
218
219         (* fill up the buffer with the appropriate fields *)
220         fillb(CCCQDE, swndno.fixp[1]);
221   {     writeln(tty,'enter the SOURCE address')};
222         readln(input);  (* skip over comment *)
223         readln(input, a);
224         fillw(a, swndno.fixp[5], swndno.fixp[6]);
225   {     writeln(tty,'enter the CLASS and OPTION')};
226         readln(input);  (* skip over comment *)
227         readln(input, a);
228         fillb(a, swndno.fixp[8]);
229   {     writeln(tty,'enter the CALLING TSAP identifier')};
230         readln(input);  (* skip over comment *)
231         readln(input, a);
232         fillb(a, swndno.oth[4]);
233         fillb(CLGTSAPPC, swndno.oth[2]);
234         fillb(CLGTSAPPL, swndno.oth[1]);
235   {     writeln(tty,'enter the CALLED TSAP identifier')};
236         readln(input);  (* skip over comment *)
237         readln(input, a);
238         fillb(a, swndno.oth[5]);
239         fillb(CLDTSAPPC, swndno.oth[3]);
240         fillb(CLDTSAPPL, swndno.oth[6]);
```

```
241   {     writeln(tty,'enter the TPDU size');}
242         readln(input);  (* skip over comment *)
243         readln(input, a);
244         fillb(a, swndno.oth[10]);
245         fillb(TPDUSZPL, swndno.oth[7]);
246         fillb(TPDUSZPC, swndno.oth[8]);
247
248   {     writeln(tty,'enter the 4 THROUGHPUT values (8 octets)');}
249         readln(input);  (* skip over comment *)
250         read(input, a);
251         fillw(a, swndno.oth[11], swndno.oth[12]);
252         read(input, a);
253         fillw(a, swndno.oth[13], swndno.oth[14]);
254         read(input, a);
255         fillw(a, swndno.oth[15], swndno.oth[16]);
256         readln(input, a);
257         fillw(a, swndno.oth[17], swndno.oth[18]);
258
259         fillb(THPC, swndno.oth[9]);
260         fillb(THPL, swndno.oth[23]);
261
262   {     writeln(tty,'enter the 4 TRANSIT DELAY values (8 octets)');}
263         readln(input);  (* skip over comment *)
264         read(input, a);
265         fillw(a, swndno.oth[27], swndno.oth[28]);
266         read(input, a);
267         fillw(a, swndno.oth[29], swndno.oth[30]);
268         read(input, a);
269         fillw(a, swndno.oth[31], swndno.oth[32]);
270         readln(input, a);
271         fillw(a, swndno.oth[33], swndno.oth[34]);
272
273         fillb(TRDLPC, swndno.oth[25]);
274         fillb(TRDLPL, swndno.oth[24]);
275       end                                                    (* tcres *)
276       else if userXerin = 'tnodr ' then                      (* tnodr *)
277       begin
278         (* fill up the buffer with the appropriate fields *)
279         fillb(DTCODE, swndno.fixp[1]);
280         fillb(NREOT, swndno.fixp[4]);
281         writeln(tty,'enter USER DATA');
282         readUdata(swndno)
283       end                                                    (* tnodr *)
```

```
284       else if userXprim = 'tdreq ' then                          (* tdreq *)
285       begin
286         (* go in the tdreq file for input *)
287         reset(input, 'TDREQ.TXT');
288         if ioresult(input) < 0 then
289         begin
290           writeln(tty, 'Error opening the file called TDREQ.TXT');
291           halt
292         end;
293
294         (* fill up the buffer with the appropriate fields *)
295         fillb(DRCODE, swndno.fixp[1]);
296   {     writeln(tty,'enter the DESTINATION address');}
297         readln(input);  (* skip over comment *)
298         readln(input, a);
299         fillw(a, swndno.fixp[3], swndno.fixp[4]);
300   {     writeln(tty,'enter the SOURCE address');}
301         readln(input);  (* skip over comment *)
302         readln(input, a);
303         fillw(a, swndno.fixp[5], swndno.fixp[6]);
304   {     writeln(tty,'enter the REASON for disconnection');}
305         readln(input);  (* skip over comment *)
306         readln(input, a);
307         fillb(a, swndno.fixp[8]);
308         writeln(tty,'enter the additional information for disconnection');
309         fillb(DVCODE, swndno.oth[2]);
310         readUdata(swndno)
311       end                                                        (* tdreq *)
312       else                                                       (* error *)
313       begin
314         writeln(tty, 'error in inputed primitive');
315         valid := false
316       end                                                        (* error *)
317       if valid then
318       begin
319         writeln(tty, 'USER A primitive successfully sent to TPRENTA');
320         tsend(swndno, wndno, id)     (*  send buffer to TPRENTA *)
321       end;
322
323     end;                                        (* DO forever *)
324
325     writeln(tty,' Task TERMA terminated')
326   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      011566
OUTERMOST DATA SIZE     000450
RESERVED STACK & HEAP   002060
COMPILATION TIME 104 SECONDS

PAS TERMA,TERMA/P42=TERMA
```

```
    1   (*$m-*)
    2   (*****************************************************************************)
    3
    4      NAME:         TINIT
    5
    6      FUNCTION:     This module is used to create a dynamic region
    7                    for task TERNA by issuing a CRRG$ memory management directive.
    8
    9
   10      USAGE:        procedure tinit(var id : integer); extern;
   11                                .
   12                                .
   13                                .
   14                    tinit(id);
   15
   16      INPUT PARAMETERS:    NONE
   17
   18      OUTPUT PARAMETERS:   id      - the region id.
   19
   20      MODULES CALLED:      (The symbol '>' designates a primitive)
   21
   22                           >halt
   23                           >writeln
   24                           crrg
   25
   26      EXIT CONDITIONS:
   27                           If there is an error detected in calling CRRG$, then
   28                           an error message identifying the error and error code
   29                           is printed onto standard I/O and the processor is then
   30                           halted by using the 'halt' primitive.
   31
   32                           The return code of the CRRG$ routine is set to greater
   33                           or equal to 0 (zero) in case of successful completion and to
   34                           less than 0 in case of unsuccessful completion.
   35
   36                           For more information on the use of CRRG$ and of its
   37                           return codes, see pages 6-36..6-38 of the RSX-11M V3.2
   38                           Executive Reference Manual.
   39
   40      REMARKS:             At the moment, there is no protection on the region.
   41                           One can easily remedy this by giving an appropriate
   42                           value to the region protection word.
   43
```

```
   44   ********************************************************************************)
   45
   46   {$I+TERMA.DEF}
```

```
 1
 2   const
 3           REQLEN     = 6;
 4           NOPTR      = 7;
 5           WNDSZ      = 3;
 6           APRNO      = 7;
 7           REGSZ      = 21;
 8           FIXPLEN    = 8;
 9           OTHLEN     = 130;
10           STRLEN     = 118;
11           CRCODE     = 224;
12           CCCODE     = 208;
13           DTCODE     = 240;
14           DRCODE     = 128;
15           DVCODE     = 224;
16           CLGTSAPPC  = 193;
17           CLGTSAPPL  = 1;
18           CLDTSAPPC  = 194;
19           CLDTSAPPL  = 1;
20           TPDUSZPC   = 192;
21           TPDUSZPL   = 1;
22           THPC       = 137;
23           THPL       = 12;
24           TRDLPC     = 136;
25           TRDLPL     = 8;
26           NREOT      = 99;
27
28   type
29           std = record
30                   fixp : packed array [1..FIXPLEN] of char;
31                   oth : packed array [1..OTHLEN] of char;
32           end;
33
34           pstd = ^std;
35
36           contrl = array[0..7] of integer;
37
38           ddd = record
39                   ok : array[0..7] of integer;
40           end;
41
42           pdd = ^ddd;
43
```

```
 44          wdbb = record  { window block format }
 45                  apr : packed array[1..2] of char;
 46                  vbadd : pstd;
 47                  siz : integer;
 48                  regid : integer;
 49                  offs : integer;
 50                  len : integer;
 51                  wstats : integer;
 52                  srbuff : pdd;
 53          end;
 54
 55          tskk = record
 56                  nam : array[1..2] of integer;
 57          end;
 58
 59          efnn = integer;
 60
 61          idss = integer;
 62
 63          request = packed array[1..REQLEN] of char;
 64
 65          wndnoo = 1..NOPTR;
 66
```

```
47
48
49  procedure tinit(var id : integer);
50
51  var
52          control  : contrl;
53          ids      : idss;
54
55  procedure crrs(var control : contrl; var ids : idss); extern (fortran);
56
57  begin
58  { writeln(tty, 'entering tinit subroutine'); }
59
60    (* create and attach to region with region name 'A1'*)
61    (* control[0]  region id returned *)
62    control[1] := REGSZ;          (* size of the region in 32-word blocks also returned  *)
63    control[2] := 2840;           (* region name (2 words in Radix-50 format)*)
64    control[3] := 0;              (*  or 0 for no-name, 'A1'                            *)
65    control[4] := 11414;          (* Name of the partition that contains the *)
66    control[5] := 0;              (* region.  (2 word in Radix-50 format     *)
67                                  (* presently it is GEN *)
68                                  (* or 0 for the partition in which the task is
69                                     running *)
70    control[6] := 35;             (* Region status word (see page 3-14) also returned *)
71                                  (* bits set:
72                                            RS.ATT = 32
73                                            RS.WRT = 2
74                                            RS.RED = 1                        *)
75
76    control[7] := 0;              (* Region protection code *)
77                                  (* everything is permitted see p 3-13 *)
78    crrs(control, ids);
79  { writeln(tty, 'id assigned to the created region = ', control[0]);
80    writeln(tty, 'size in 32-W blocks of the attached region = ', control[1]);
81    writeln(tty, 'region status word = ', control[6]);
82    writeln(tty, 'ids = ', ids);    }
83    if ids < 0 then
84    begin
85       writeln(tty, 'error on crrs,  error code = ', ids);
86       halt
87    end;
88
89    id := control[0]  (* stick region id in appropriate return field *)
90
```

```
   91     { writeln(tty, 'leaving tinit subroutine'); }
   92   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      000650
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000542
COMPILATION TIME 13 SECONDS

PAS TINIT,TINIT/P42-TINIT
```

```
    1   (*****************************************************************************
    2
    3       NAME:         TPRENTA (MAIN)
    4
    5       FUNCTION:     To serve as Transport Protocol Entity for USER 'A'.
    6
    7       USAGE:        from MCR : 'RUN TPRENTA /TASK=TPRENA /INC=100.'
    8
    9       INPUT PARAMETERS:     NONE
   10
   11       OUTPUT PARAMETERS:    NONE
   12
   13       MODULES CALLED:       (The symbol '>' designates a primitive)
   14
   15                             >chr
   16                             >new
   17                             >ord
   18                             >write
   19                             >writeln
   20                             buffclear
   21                             buffgo
   22                             buffwait
   23                             initial
   24                             insptpdu
   25                             preptpdu
   26                             recca
   27                             sendd
   28                             werra
   29                             wterma
   30
   31       EXIT CONDITIONS:
   32                             This program has been designed to run indefinitely until
   33                             the task into which it runs is specifically aborted with
   34                             the MCR command 'ABORT TPRENA'.
   35
   36                             If an error is detected by one of the routines called,
   37                             then the routine itself will print an error message and
   38                             halt the processor by using the 'halt' primitive.
   39
   40       REMARKS:
   41                             The format of the TPDUs is different from the standard
   42                             format since the PDP-11 swaps the HI and LO bytes of a
   43                             word in memory.  So internally speaking (in memory) the
```

```
   44                          IFIOUs are according to standards.
   45
   46   ********************************************************************************)
   47
   48   program TPRENTA(tty);
   49
   50   {$I+TPRENTA.DEF}
```

```
    1
    2   const                          { GLOBAL CONSTANTS }
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           STRLEN     = 118;
    6           NETREQLEN  = 6;
    7           CRCODE     = 224;
    8           CCCODE     = 208;
    9           DTCODE     = 240;
   10           DRCODE     = 128;
   11           NDRCODE    = 120;
   12           ERCODE     = 112;
   13           APRNO      = 7;
   14           WNDSZ      = 3;
   15           REGSZ      = 21;   { NOPTR times WNDSZ }
   16           NOPTR      = 7;
   17           DUMLEN     = 10;
   18           DUMMLEN    = 5;
   19           NODATFLG   = 20;
   20           IVCODE     = 193;
   21           DVCODE     = 224;
   22           ERRLEN     = 128;
   23           GCDEST     = 2222;
   24           GCSOUR     = 1111;
   25           TPDUSZPL   = 1;
   26           TPDUSZPC   = 192;
   27           THPC       = 137;
   28           GCTPDU     = 128;
   29           GCTHR1     = 1;
   30           GCTHR2     = 2;
   31           GCTHR3     = 3;
   32           GCTHR4     = 4;
   33           THPL       = 12;
   34           TRDLPL     = 8;
   35           TRDLPC     = 136;
   36           GCTRD1     = 7;
   37           GCTRD2     = 8;
   38           GCTRD3     = 9;
   39           GCTRD4     = 10;
   40           NREOT      = 99;
   41           GCRESO     = 11;
   42           UNDEF      = 99;
   43
```

```
44
45  type                            { GLOBAL TYPES }
46          std = record            { TPDUs }
47                  fixp : packed array [1..FIXPLEN] of char;
48                  oth  : packed array [1..OTHLEN] of char;
49          end;
50
51          pstd = ^std;
52
53          contrl = array[0..7] of integer; { format for CRRG$ }
54
55          netXreq   = packed array[1..NETREQLEN] of char;
56
57          ddd = record
58                  req : netXreq;
59                  dum : array[1..DUMMLEN] of integer;
60          end;
61
62          pdd = ^ddd;
63
64          tskk = record
65                  nam : array [1..2] of integer;
66          end;
67
68          rbuff = record { format for RCVD$ }
69                  nam : tskk;
70                  req : netXreq;
71                  dum : array [1..DUMLEN] of integer;
72          end;
73
74          buff = record  { format for SDAT$ }
75                  req : netXreq;
76                  dum : array [1..DUMLEN] of integer;
77          end;
78
79          rddd = record
80                  ok : array[1..2] of integer;
81                  req : netXreq;
82                  dum : array[1..DUMMLEN] of integer;
83          end;
84
85          rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rregid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff :prpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  regid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmsg = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
  130
  131           errr = integer;
```

```
51
52
53
54   (*  FORMAT OF TPDUs      [record of type 'std']
55
56     Connect request:
57           fixp[1]      - CRCDT
58           fixp[2]      - LI
59           fixp[3]      - not used
60           fixp[4]      - not used
61           fixp[5]      - SRCREF (LO)
62           fixp[6]      - SRCREF (HI)
63           fixp[7]      - not used
64           fixp[8]      - CLASSXOP
65
66           oth[1]       - CLGTSAPPL
67           oth[2]       - CLGTSAPPC
68           oth[3]       - CLDTSAPPC
69           oth[4]       - CLGTSAPVL
70           oth[5]       - CLDTSAPVL
71           oth[6]       - CLDTSAPPL
72           oth[7]       - TPDUSZPL
73           oth[8]       - TPDUSZPC
74           oth[9]       - THPC
75           oth[10]      - TPDUSZVL
76           oth[11..22]  - THVL     oth[11..12]  - THROUGHPUT 1
77                                   oth[13..14]  - THROUGHPUT 2
78                                   oth[15..16]  - THROUGHPUT 3
79                                   oth[17..18]  - THROUGHPUT 4
80                                   oth[19..22]  - not used
81           oth[23]      - THPL
82           oth[24]      - TRDLPL
83           oth[25]      - TRDLPC
84           oth[26]      - not used
85           oth[27..34]  - TRDLVL
86           oth[35]      - not used
87           oth[36..130] - not used
88
89     Connect confirm:
90           fixp[1]      - CCCDT
91           fixp[2]      - LI
92           fixp[3]      - not used
93           fixp[4]      - not used
94           fixp[5]      - SRCREF (LO)
```

```
 95          fixp[6]     - SRCREF (HI)
 96          fixp[7]     - not used
 97          fixp[8]     - CLASSXOP
 98
 99          oth[1]      - CLGTSAPPL
100          oth[2]      - CLGTSAPPC
101          oth[3]      - CLDTSAPPC
102          oth[4]      - CLGTSAPVL
103          oth[5]      - CLDTSAPVL
104          oth[6]      - CLDTSAPPL
105          oth[7]      - TPDUSZPL
106          oth[8]      - TPDUSZPC
107          oth[9]      - THPC
108          oth[10]     - TPDUSZVL
109          oth[11..22] - THVL    oth[11..12] - THROUGHPUT 1
110                                oth[13..14] - THROUGHPUT 2
111                                oth[15..16] - THROUGHPUT 3
112                                oth[17..18] - THROUGHPUT 4
113                                oth[19..22] - not used
114          oth[23]     - THPL
115          oth[24]     - TRDLPL
116          oth[25]     - TRDLPC
117          oth[26]     - not used
118          oth[27..34] - TRDLVL
119          oth[35]     - not used
120          oth[36..130]- not used
121
122     Data transfer:
123          fixp[1]     - DTC
124          fixp[2]     - LI
125          fixp[3]     - not used
126          fixp[4]     - NREOT
127          fixp[5..8]  - not used
128
129          oth[1..120] - USER DATA  oth[1]     - string len
130                                   oth[2]     - not used
131                                   oth[3..120]- string
132          oth[121..130] - not used
133
134     Error:
135          fixp[1]     - ERC
136          fixp[2]     - LI
137          fixp[3]     - DSTREF (LO)
```

```
138             fixp[4]     - DSTREF (HI)
139             fixp[5]     - not used
140             fixp[6]     - REJCSE
141             fixp[7..8]  - not used
142
143             oth[1]      - not used
144             oth[2]      - INVTPDUPC
145             oth[3..130] - INVTPDU
146
147     Disconnect Request:
148             fixp[1]     - DRC
149             fixp[2]     - LI
150             fixp[3]     - DSTREF (LO)
151             fixp[4]     - DSTREF (HI)
152             fixp[5]     - SRCREF (LO)
153             fixp[6]     - SRCREF (HI)
154             fixp[7]     - not used
155             fixp[8]     - REASON
156
157             oth[1]      - DVL
158             oth[2]      - DVC
159             oth[3..120] - DVVLL
160             oth[121..130] - not used
161
162                                                    *)
163
164   var
165             dstref      : arri;
166             err         : errri;
167             id          : integeri;
168             messXterma  : booleani;
169             netXcon     : booleani;
170             nXreq       : netXreqi;
171             openXsts    : opXstsi;
172             rtrwnd      : rstdi;
173             queue       : booleani;
174             reason      : integeri;
175             rjcse       : integeri;
176             srcref      : arri;
177             stXprenta   : statei;
178             valid       : booleani;
179             wndno       : wndnooi;
180
```

```
181
182   procedure buffclear(efn : efnn); external;
183
184   procedure buffso(efn : efnn); external;
185
186   procedure buffwait(efn : efnn); external;
187
188   procedure initial(var id : integer); external;
189
190   procedure insptpdu(ptrwnd : pstd;
191                      var valid : boolean;
192                      var rjcse : integer;
193                      var err : errr); external;
194
195   procedure preptpdu(ptrwnd : pstd;
196                      var dstref : arr;
197                      var srcref : arr;
198                      rjcse : integer;
199                      err : errr;
200                      cod : integer); external;
201
202   procedure recca(var ptrwnd : pstd;
203                   var nXreq : netXreq;
204                   var queue : boolean;
205                   var messXterma : boolean); external;
206
207   procedure sendd(ptrwnd  : pstd;
208                   var wndno : wndnqo;
209                   nXreq : netXreq;
210                   id : integer;
211                   bol : boolean); external;
212
213   procedure werra(ptrwnd : pstd;
214                   rjcse : integer;
215                   mes : wmessXerr); external;
216
217   procedure wterma(ptrwnd  : pstd;
218                    mes : wmessXsts); external;
219
220
221   begin  (* MAIN *)
222
223     (* for display purposes on VT100 terminal *)
```

```
 224   ( writeln(tty, chr(27),'[2J',
 225                  chr(27), '[0;0f', chr(27), '[0m',
 226                  chr(27), '[7;24r', chr(27), '[7m', chr(27), '[A');
 227     writeln(tty, '*****************************************************************************');
 228     writeln(tty, '*                                                                           *');
 229     writeln(tty, '*                T A S K   T P R E N T A  activated                          *');
 230     writeln(tty, '*                                                                           *');
 231     writeln(tty, '*****************************************************************************');
 232     writeln(tty, chr(27), '[0m');)
 233
 234     wndno := 1;           (* initialize sending window *)
 235     rjcse := 0;           (* initialize tpdu rejection cause *)
 236     stXprenta := CLOSED;  (* initialize state of TPRENTA *)
 237     openXsts := closeds;  (* initialize open status of TPRENTA *)
 238     netXcon := false;     (* initialize status of network connection *)
 239     new(ptrwnd);          (* create dynamic space for TPDU *)
 240
 241     initial(id);          (* create a dynamic region with a name of 'A2' *)
 242
 243
 244   (* make sure the screen is clear for display purposes *)
 245   writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
 246   writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
 247   writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
 248
 249     while(1 = 1) do         (* DO this forever *)
 250     begin
 251
 252       (* print out the state of the transport protocol entity *)
 253       write(tty, chr(27), '[7;20f', 'STATE < ');
 254       if (ord(stXprenta) = 0) then
 255         writeln(tty, 'CLOSED >')
 256       else if (ord(stXprenta) = 1) then
 257         writeln(tty, 'OPEN   >')
 258       else if (ord(stXprenta) = 2) then
 259         writeln(tty, 'WFCC   ')
 260       else if (ord(stXprenta) = 3) then
 261         writeln(tty, 'WFTRESP')
 262       else
 263         writeln(tty, 'WFNC   ');
 264
 265       (* receive packet from receive by reference queue or by task queue*)
 266       recca(ptrwnd, nXreu, queue, messXterma);
```

```
267
268       (* print out the primitive or/and the type of TPDU received *)
269       if queue then
270       begin
271         writeln(tty, chr(27), '[8;20f', 'RECEIVED : ', nX,eu);
272         writeln(tty, chr(27), '[9;20f' , 'TPDU  : NONE          ')
273       end
274       else if messXterma then
275       begin
276         write(tty, chr(27), '[8;20f', 'RECEIVED : ')
277         if ord(ptrwnd^.fixp[1]) = CRCODE then
278           writeln(tty, 'tcrea')
279         else if ord(ptrwnd^.fixp[1]) = CCCODE then
280           writeln(tty, 'tcres')
281         else if ord(ptrwnd^.fixp[1]) = DTCODE then
282           writeln(tty, 'tnodr')
283         else if ord(ptrwnd^.fixp[1]) = DRCODE then
284           writeln(tty, 'tdrea')
285         else
286           writeln(tty, '      ');
287         writeln(tty, chr(27), '[9;20f' , 'TPDU  : NONE          ')
288       end
289       else
290       begin
291         writeln(tty, chr(27), '[8;20f', 'RECEIVED :             ');
292         write(tty, chr(27), '[9;20f' , 'TPDU  :')
293         if ord(ptrwnd^.fixp[1]) = CRCODE then
294           writeln(tty, ' "CR"')
295         else if ord(ptrwnd^.fixp[1]) = CCCODE then
296           writeln(tty, ' "CC"')
297         else if ord(ptrwnd^.fixp[1]) = DTCODE then
298           writeln(tty, ' "DT"')
299         else if ord(ptrwnd^.fixp[1]) = DRCODE then
300           writeln(tty, ' "DR"')
301         else if ord(ptrwnd^.fixp[1]) = ERCODE then
302           writeln(tty, ' "ER"')
303         else
304           writeln(tty, ' "XX"')
305       end;
306       writeln(tty, chr(27), '[23;1f');
307
308       if messXterma then                    (* message from terma ? *)
309       begin
```

```
310
311         case stXprenta of                                       (* check state of TPRENTA *)
312
313         CLOSED:
314           begin
315             if (ptrwnd^.fixpC1] = chr(CRCODE)) then
316             begin
317               if (not(netXcon)) then                            (* P2 *)
318               begin
319                 openXsts := inprogress;
320                 preptpdu(ptrwnd, dstref, srcref, rjcse, err, CRCODE);
321                 sendd(ptrwnd, wndno, 'ncreq ', id, true);       (* send nXreq to NTNRSDA *)
322                 stXprenta := WFNC
323               end                                               (* P2 *)
324               else
325               begin
326                 if ((netXcon) and (openXsts = inprogress)) then (* P4 *)
327                   stXprenta := WFNC
328                 else
329                 begin
330                   if ((netXcon) and (openXsts = opens)) then    (* P3 *)
331                   begin
332                     stXprenta := WFCC;
333                     preptpdu(ptrwnd, dstref, srcref, rjcse, err, CRCODE);
334                     sendd(ptrwnd, wndno, nXreq, id, false)      (* send  to NTRNSDA *)
335                   end                                           (* P3 *)
336                   else
337                     werra(ptrwnd, rjcse, five)                  (* tcreq unacceptable P0 *)
338                 end
339               end
340             end                                                 (* tcreq *)
341             else
342               werra(ptrwnd, rjcse, six)
343           end;                                                  (* closed state *)
344
345         OPEN:
346           begin
347             if (ptrwnd^.fixpC1] = chr(DTCODE)) then
348             begin
349               preptpdu(ptrwnd, dstref, srcref, rjcse, err, DTCODE);
350               sendd(ptrwnd, wndno, nXreq, id, false);           (* send  to NTRNSDA *)
351             end
352             else
```

```
353             begin
354               if (ptrwnd^.fixp[1] = chr(DRCODE)) then
355               begin
356                 netXcon := false;
357                 openXsts :=closeds;
358                 sendd(ptrwnd, wndno, 'ndreq ', id, true); {     (* send nXreq to NTRNSDA *)
359                 stXprenta := CLOSED
360               end
361               else
362                 werra(ptrwnd, rjcse, seven);
363             end
364           end;                                                (* OPEN state *)
365
366         WFCC:
367           begin
368             if (ptrwnd^.fixp[1] = chr(DRCODE)) then
369             begin
370               netXcon := false;
371               openXsts :=closeds;
372               sendd(ptrwnd, wndno, 'ndreq ', id, true);       (* send nXreq to NTRNSDA *)
373               stXprenta := CLOSED
374             end
375             else
376               werra(ptrwnd, rjcse, eight)
377           end;                                                (* WFCC state *)
378
379         WFTRESP:
380           begin
381             if (ptrwnd^.fixp[1] = chr(CCCODE)) then
382             begin
383               preptpdu(ptrwnd, dstref, srcref, rjcse, err, CCCODE);
384               sendd(ptrwnd, wndno, nXreq, id, false);         (* send  to NTRNSDA *)
385               stXprenta := OPEN
386             end                                               (* tcres *)
387             else
388             begin
389               if (ptrwnd^.fixp[1] = chr(DRCODE)) then
390               begin
391                 preptpdu(ptrwnd, dstref, srcref, rjcse, err, DRCODE);
392                 sendd(ptrwnd, wndno, nXreq, id, false);       (* send  to NTRNSDA *)
393                 stXprenta := CLOSED
394               end
395               else
```

```
396                 werra(ptrwnd, rdcse, nine);
397               end
398             end;                                          (* WFTRESP state *)
399
400           WFNC:
401             begin
402               if(ptrwnd^.fixp[1] <> chr(DRCODE)) then
403               begin
404                 netXcon := false;
405                 openXsts :=closedst;
406                 sendd(ptrwnd, wndno, 'ndreq ', id, true);     (* send nXreq to NTRNSDA *)
407                 stXprenta := CLOSED
408               end
409               else
410               begin
411                 werra(ptrwnd, rdcse, ten);
412               end
413             end                                           (* WFNC state *)
414           end                                             (* CASE *)
415         end                                               (* if message from TERMA *)
416         else                                              (* message from network service
417                                                             provider *)
418         begin
419           case stXprenta of
420
421           CLOSED :
422             begin
423               if (queue) then
424               begin
425                 if (nXreq = 'ncind ') then
426                 begin
427                   netXcon := true;
428                   openXsts := opens;
429                   sendd(ptrwnd, wndno, 'ncres ', id, true)    (* send nXreq to NTRNSDA *)
430                 end
431                 else if (nXreq = 'ndind ') then
432                 begin
433                   openXsts := closedst;
434                   netXcon := false
435                 end
436               end                                         (* not queue *)
437               else                                        (* if queue *)
438               begin
```

```
439             if (ptrwnd^.fixp[1] = chr(CRCODE)) then
440             begin
441               inspthdu(ptrwnd, valid, rjcse, err);
442               if valid then
443               begin
444                 stXprenta := WFTRESP;
445                 dstref[1]:= ptrwnd^.fixp[5];                (* set the initiator's *)
446                 dstref[2] := ptrwnd^.fixp[6];               (* address             *)
447                 wterma(ptrwnd, one)
448               end
449               else
450               begin
451                 werra(ptrwnd, rjcse, nineteen);
452                 prepthdu(ptrwnd, dstref, srcref, rjcse, err, NDRCODE);
453                 sendd(ptrwnd, wndno, nXreq, id, false)
454               end
455             end
456             else
457               werra(ptrwnd, rjcse, fourteen)
458           end
459
460         end;                                                (* CLOSED *)
461
462       OPEN :
463         begin
464           if (queue) then
465           begin
466             if (nXreq = 'nrind ') then
467             begin
468               wterma(ptrwnd, zero);                         (* give TDIND to the TS user A *)
469               netXcon := false;
470               openXsts :=closedst;
471               sendd(ptrwnd, wndno, 'ndreq ', id, true);     (* send nXreq to NTRNSDA *).
472                                                             (* [1] *)
473               if (not netXcon) then
474                 sendd(ptrwnd, wndno, 'nrres ', id, true);   (* send nXreq to NTRNSDA *)
475                                                             (* [5] *)
476               stXprenta := CLOSED
477             end                                             (* nrind *)
478             else
479             begin
480               if (nXreq = 'ndind ') then
481               begin
```

```
482                 netXcon := false;
483                 openXsts :=closedS;
484                 wterma(ptrwnd, zero);                    (* give TDIND to TS user *)
485                 stXprenta := CLOSED
486               end
487             end
488           end                                            (* if queue *)
489           else                                           (* if not queue *)
490           begin
491             if (ptrwnd^.fixp[1] = chr(DTCODE)) then
492             begin
493               insptpdu(ptrwnd, valid, rjcse, err);
494               if valid then
495                 wterma(ptrwnd, three)                    (* give TNODI to user R *)
496               else
497               begin
498                 werra(ptrwnd, rjcse, seventeen);
499                 preptpdu(ptrwnd, dstref, srcref, rjcse, err, ERCODE);
500                 sendd(ptrwnd, wndno, nXreq, id, false)   (* send ER TPDU to NTRNSDA *)
501               end
502             end
503             else
504             begin
505               if (ptrwnd^.fixp[1] = chr(DRCODE)) then
506               begin
507                 netXcon := false;
508                 openXsts :=closedS;
509                 sendd(ptrwnd, wndno, 'ndreq ', id, true);   (* send nXreq to NTRNSDA *)
510                 stXprenta := CLOSED
511               end
512               else
513               begin
514                 if (ptrwnd^.fixp[1] = chr(ERCODE)) then
515                 begin
516                   werra(ptrwnd, rjcse, eleven);
517                   netXcon := false;
518                   openXsts :=closedS;
519                   sendd(ptrwnd, wndno, 'ndreq ', id, true);   (* send nXreq to NTRNSDA *)
520                   stXprenta := CLOSED
521                 end                                      (* er *)
522               end                                        (* else *)
523             end                                          (* else *)
524           end                                            (* not queue *)
```

```
525           end;                                                  (* OPEN *)
526
527         WFNC :
528           begin
529             if (queue) then
530             begin
531               if (nXreq = 'nccon ') then
532               begin
533                 netXcon := true;
534                 openXsts := opens;
535                 sendd(ptrwnd, wndno, nXreq, id, false);         (* send  to NTRNSDA *)
536                 stXprenta := WFCC
537               end
538               else
539               begin
540                 if (nXreq = 'nrind ') then
541                 begin
542                   wterma(ptrwnd, zero);                         (* give TDIND to the TS user A *)
543                   netXcon := false;
544                   openXsts :=closeds;
545                   sendd(ptrwnd, wndno, 'ndreq ', id, true);     (* send nXreq to NTRNSDA *)
546                                                                 (* [1] *)
547                   if (not netXcon) then
548                     sendd(ptrwnd, wndno, nXreq, id, true);      (* send nXreq to NTRNSDA *)
549                                                                 (* [5] *)
550                   stXprenta := CLOSED
551                 end                                             (* nrind *)
552                 else
553                 begin
554                   if (nXreq = 'ndind ') then
555                   begin
556                     netXcon := false;
557                     openXsts :=closeds;
558                     wterma(ptrwnd, zero);                       (* give TDIND to TS user A *)
559                     stXprenta := CLOSED
560                   end
561                 end
562               end
563             end                                                 (* if queue *)
564             else                                                (* if not queue *)
565               werra(ptrwnd, rjcse, fifteen)
566           end;                                                  (* WFNC *)
567
```

```
568         WFCC :
569          begin
570            if (queue) then
571            begin
572              if (nXreq = 'nrind ') then
573              begin
574                wterma(ptrwnd, zero);                    (* give TDIND to user A *)
575                netXcon := false;
576                openXsts :=closeds;
577                sendd(ptrwnd, wndno, 'ndreq ', id, true);   (* send nXreq to NTRNSDA *)
578                                                         (* [1] *)
579                if (not netXcon) then
580                  sendd(ptrwnd, wndno, 'nrres ', id, true);  (* send nXreq to NTRNSDA *)
581                                                         (* [5] *)
582                stXprenta := CLOSED
583              end                                        (* nrind *)
584              else
585              begin
586                if (nXreq = 'mdind ') then
587                begin
588                  wterma(ptrwnd, zero);                  (* give TDIND user A *)
589                  netXcon := false;
590                  openXsts :=closeds;
591                  stXprenta := CLOSED
592                end
593              end
594            end                                          (* if queue *)
595            else                                         (* if not queue *)
596            begin
597              if (ptrwnd^.fixp[1] = chr(CCCODE)) then
598              begin
599                insetpdu(ptrwnd, valid, rjcse, err);     (* check validity *)
600                if valid then                            (* i.e cc acceptable p8 *)
601                begin
602                  wterma(ptrwnd, two);                   (* give TCCON to user A *)
603                  stXprenta := OPEN
604                end
605                else                                     (* invalid cc *)
606                begin
607                  werra(ptrwnd, rjcse, twelve);
608                  netXcon := false;
609                  openXsts :=closeds;
610                  sendd(ptrwnd, wndno, 'ndreq ', id, true);   (* send nXreq to NTRNSDA *)
```

```
611                 stXprenta := CLOSED
612               end
613             end                                          (* cc *)
614             else
615             begin
616               if (ptrwnd^.fixp[1] - chr(ERCODE)) then
617               begin
618                 werra(ptrwnd, rjcse, thirteen);          (* with reasons etc.*)
619                 netXcon := false;
620                 openXsts :=closeds;
621                 sendd(ptrwnd, wndno, 'ndreq ', id, true);  (* send nXreq to NTRNSDA *)
622                 stXprenta := CLOSED
623               end                                        (* er *)
624               else
625               begin
626                 if (ptrwnd^.fixp[1] - chr(DRCODE)) then
627                 begin
628                   insptpdu(ptrwnd, valid, rjcse, err);
629                   if valid then
630                   begin
631                     wterma(ptrwnd, four);                (* give TDIND to user A *)
632                     netXcon := false;
633                     openXsts :=closeds;
634                     sendd(ptrwnd, wndno, 'ndreq ', id, true); (* send nXreq to NTRNSDA *)
635                     stXprenta := CLOSED
636                   end
637                   else
638                   begin
639                     werra(ptrwnd, rjcse, eighteen);
640                     preptpdu(ptrwnd, dstref, srcref, rjcse, err, ERCODE);
641                     sendd(ptrwnd, wndno, nXreq, id, false);  (* send ER TPDU to NTRNSDA *)
642                   end
643                 end                                      (* dr *)
644               end
645             end
646           end                                            (* if not queue *)
647         end;                                             (* WFCC *)
648
649       WFTRESP :
650         begin
651           if (queue) then
652           begin
653             if (nXreq = 'nrind ') then
```

```
654                 begin
655                   wterma(ptrwnd, zero);'                    (* give TDIND to user A *)
656                   netXcon := false;
657                   openXsts :=closeds;
658                   sendd(ptrwnd, wndno, 'ndreq ', id, true);  (* send nXreq to NTRNSDA *)
659                                                              (* [1] *)
660                   if (not netXcon) then
661                     sendd(ptrwnd, wndno, 'nrres ', id, true); (* send nXreq to NTRNSDA *)
662                                                              (* [5] *)
663                   stXprenta := CLOSED
664                 end                                          (* nrind *)
665                 else
666                 begin
667                   if (nXreq = 'ndind ') then
668                   begin
669                     netXcon := false;
670                     openXsts :=closeds;
671                     wterma(ptrwnd, zero);                    (* give TDIND to user A *)
672                     stXprenta := CLOSED
673                   end
674                 end
675               end                                            (* queue *)
676               else                                           (* not queue *)
677                 werra(ptrwnd, rjcse, sixteen)
678             end                                              (* WFTRESP *)
679           end                                                (* case *)
680         end                                                  (* end of else message *)
681
682     end;                                                     (* end of DO forever *)
683
684     writeln(tty, 'TPRENTA TASK TERMINATED.....')
685   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      020710
OUTERMOST DATA SIZE     000044
RESERVED STACK & HEAP   001072
COMPILATION TIME 217 SECONDS

PAS TPRENTA,TPRENTA/P42=TPRENTA
```

```
    1   (*$m-*)
    2   (*****************************************************************************)
    3
    4      NAME:         TSEND
    5
    6      FUNCTION:     To send a buffer of information to task TERENTA, by using the
    7                    send by reference directive SREF$.
    8
    9      USAGE:        procedure tsend(swndno : sysdi
   10                                   var wndno : windoo;
   11                                   id : integer); extern;
   12                                   .
   13                                   .
   14                                   .
   15                   sendd(swndno, wndno, id);
   16
   17      INPUT PARAMETERS:    swndno  - the buffer to send by reference.
   18                           wndno   - the window number to be used(1 to 7).
   19                           id      - the region id.
   20
   21      OUTPUT PARAMETERS:   wndno   - the new window number.
   22
   23      MODULES CALLED:      (The symbol '>' designates a primitive)
   24
   25                           >halt
   26                           >writeln
   27                           craw
   28                           sref
   29
   30      EXIT CONDITIONS:
   31                           If there are any errors detected in calling CRAW$ or
   32                           SREF$, then an error message identifying the error
   33                           and error code is printed onto standard I/O and the
   34                           processor is then halted by using the 'halt' primitive.
   35
   36                           The return code of both these system directives, is set
   37                           to greater or equal to 0(zero) in case of successful
   38                           completion and to less than 0 in case of unsuccessful
   39                           completion.
   40
   41                           For more information on the use of CRAW$ and SREF$ and
   42                           their return codes, see pages 6-31..6-34 and 6-138..6-140
   43                           of the RSX-11M V3.2 Executive Reference Manual.
```

```
44
45      REMARKS:
46
47                          The method used in this module involves putting information
48                          in the region windows and sending these by reference.
49                          Each call to this routine involves sending the next
50                          window in line (eg. the next window in the region or
51                          window number one if the last window(ie. no 7) was used
52                          on the previous call).  This strategy guarantees that at
53                          least 7 'SENDDs' can be done before any of the data sets
54                          overwritten by new data.
55  ********************************************************************************)
56
57  {$I+TERMA.DEF}
```

```
1
2   const
3           REQLEN     = 6;
4           NOPTR      = 7;
5           WNDSZ      = 3;
6           APRNO      = 7;
7           REGSZ      = 21;
8           FIXPLEN    = 8;
9           OTHLEN     = 130;
10          STRLEN     = 118;
11          CRCODE     = 224;
12          CCCODE     = 206;
13          DTCODE     = 240;
14          DRCODE     = 128;
15          DVCODE     = 224;
16          CLGTSAPPC  = 193;
17          CLGTSAPPL  = 1;
18          CLDTSAPPC  = 194;
19          CLDTSAPPL  = 1;
20          TPDUSZPC   = 192;
21          TPDUSZPL   = 1;
22          THPC       = 137;
23          THPL       = 12;
24          TRDLPC     = 136;
25          TRDLPL     = 8;
26          NREOT      = 99;
27
28  type
29          std = record
30                  fixp : packed array [1..FIXPLEN] of char;
31                  oth : packed array [1..OTHLEN] of char;
32          end;
33
34          pstd = ^std;
35
36          contrl = array[0..7] of integer;
37
38          ddd = record
39                  ok : array[0..7] of integer;
40          end;
41
42          pdd = ^ddd;
43
```

```
 44          wdbb = record  { window block format }
 45                  arr : packed array[1..2] of char;
 46                  vbadd : pstd;
 47                  siz : integer;
 48                  regid : integer;
 49                  offs : integer;
 50                  len : integer;
 51                  wstats : integer;
 52                  srbuff : pdd;
 53          end;
 54
 55          tskk = record
 56                  nam : array[1..2] of integer;
 57          end;
 58
 59          efnn = integer;
 60
 61          idss = integer;
 62
 63          request = packed array[1..REQLEN] of char;
 64
 65          wndnoo = 1..NOPTR;
 66
```

```
 58
 59
 60  procedure tsend(swndno : std;
 61                  var wndno : wndnoo;
 62                  id : integer);
 63
 64  var
 65          efn     : efnn;
 66          ids     : idss;
 67          isrb    : ddd;
 68          swdb    : wdbb;
 69          tsk     : tskk;
 70          wdb     : wdbb;
 71
 72  procedure craw(var wdb : wdbb; var ids : idss); extern(fortran);
 73
 74  procedure sref(var tsk: tskk;
 75                 var efn : efnn;
 76                 var swdb : wdbb;
 77                 var isrb : ddd;
 78                 var ids : idss); extern(fortran);
 79  begin
 80
 81  { writeln(tty, 'entering tsend procedure');
 82    writeln(tty, 'window number is ', wndno);}
 83    (* set up task name = 'TPRENA' *)
 84    tsk.nam[1] := 32658;
 85    tsk.nam[2] := 8561;
 86
 87    (* create a send window in the region *)
 88    wdb.apr[2] := chr(APRNO);       (* Active page register *)
 89    wdb.siz := WNDSZ;               (* the size of the window in 32-word blocks *)
 90    wdb.regid := id;                (* the region id *)
 91    wdb.offs := (wndno-1)*WNDSZ;    (* the offset in region *)
 92    wdb.len := WNDSZ;               (* the lenght to map *)
 93    wdb.wstats := 386;              (* window status word *)
 94                                    (* bits set :
 95                                           WS.64B - 256
 96                                           WS.MAP - 128
 97                                           WS.WRT - 2          *)
 98    craw(wdb, ids);
 99  { writeln(tty, 'window created ids =', ids);
100    writeln(tty, 'Id assigned to the window - ', ord(wdb.apr[1]));
101    writeln(tty, 'virtual base address = ', ord(wdb.vbadd));
```

```
  writeln(tty, 'lenght actually mapped in 32-word blocks -', wdb.len);
  writeln(tty, 'window status word = ', wdb.wstats); }
  if ids < 0 then
  begin
   writeln(tty, 'error in using crew, error code = ', ids);
    halt
  end;

  (* stick buffer into the send window *)
  wdb.vbadd^.fixp := swndno.fixp;
  wdb.vbadd^.oth := swndno.oth;

  (* prepare window definition block to send by reference *)
  swdb.regid := id;              (* id of the region to be sent by reference *)
  swdb.offs := (wndno - 1) * WNDSZ;(* offset in region (32W blocks) *)
  swdb.len := WNDSZ;             (* length to map (32W blocks) *)
  swdb.wstats := 3;              (* window status word *)
                                 (* receiver access permission same as sender *)
                                 (* bits set:
                                        WS.WRT = 2
                                        WS.RED = 1      *)
  sref(tsk, efn, swdb, isrb, ids);
{ writeln(tty, 'ids for sref is = ', ids);
  writeln(tty, 'code is =', ord(wdb.vbadd^.fixp[1]));}
  if ids < 0 then
  begin
     writeln(tty, ' error in using sref,  error code = ',ids);
     halt
  end;

  (* change window for next call *)
  if wndno = NOPTR then
    wndno := 1
  else
    wndno := wndno + 1

{ writeln(tty,' leaving tsend procedure') }
end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      001326
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000610
COMPILATION TIME 23 SECONDS
```

PAS TSEND,TSEND/P42-TSEND

```
(*$m-*)
(*****************************************************************************

   NAME:        VAL

   FUNCTION:    This function takes two characters c1(LO) and c2(HI) and
                returns the integer that the combination of HI-LO forms.

   USAGE:       i  : integer;

                function val(c1 : char; c2 : char) : integer ; extern;

                i := val(c1, c2);

   INPUT PARAMETERS:    c1      - the LO character.
                        c2      - the HI character.

   OUTPUT PARAMETERS:   (returned)- the integer formed by HI-LO.

   MODULES CALLED:      (The symbol '>' designates a primitive)

                        >halt
                        >writeln

   EXIT CONDITIONS:
                        Normal exit conditions (top_down)
   REMARKS:
                        One should be aware that the hi and low bytes of a word
                        are positioned as LO-HI in memory on the PDP-11.

*****************************************************************************)

function val(c1 : char; c2 : char) : integer;

type
        trick = record
                (* variant part *)
                case bol : boolean of
                        false : ( w : integer);
                        true  : ( c : packed array [1..2] of char)
                end;
```

```
   44   var
   45           tt     : trick;
   46
   47   begin
   48   { writeln(tty, 'entering val function')}
   49
   50     tt.bol := true;
   51     tt.c[1] := c1;   (* put the characters in the variant record *)
   52     tt.c[2] := c2;
   53     tt.bol := false;
   54     val := tt.w      (* assisne val the value of the character combination *)
   55
   56   { writeln(tty, 'leaving val subroutine')}
   57   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      000230
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000524
COMPILATION TIME 5 SECONDS

PAS VAL,VAL/P42=VAL
```

```
PASCAL  PDP-11  VERSION  6.07       1908-08-04      04:51:57          PAGE   1
WERRA.PAS                             .MAIN.

    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:         WERRA
    5
    6      FUNCTION:     Write an appropriate error message on User A's terminal
    7                    due to an error situation in TPRENTA.
    8
    9      USAGE:        procedure werra(ptrwnd : pstd;
   10                                    rjcse : integer;
   11                                    mes : wmessXerr); extern;
   12                                .
   13                                .
   14                                .
   15                    werra(ptrwnd, rjcse, mes);
   16
   17      INPUT PARAMETERS:    ptrwnd  - pointer to received TPDU.
   18                           rjcse   - cause for rejection of TPDU.
   19                           mes     - the error message number.
   20
   21      OUTPUT PARAMETERS:   NONE
   22
   23      MODULES CALLED:      (The symbol '>' designates a primitive)
   24
   25                           >ord
   26                           >writeln
   27
   28      EXIT CONDITIONS:
   29                           Normal exit conditions.
   30
   31      REMARKS:
   32                           The TPRENTA module should be studied for a better
   33                           understanding of the error encountered.
   34
   35  *****************************************************************************)
   36
   37  {$I+TPRENTA.DEF}
```

```
 1
 2   const                          { GLOBAL CONSTANTS }
 3           FIXPLEN    = 8;
 4           OTHLEN     = 130;
 5           STRLEN     = 118;
 6           NETREQLEN  = 6;
 7           CRCODE     = 224;
 8           CCCODE     = 208;
 9           DTCODE     = 240;
10           DRCODE     = 128;
11           NDRCODE    = 120;
12           ERCODE     = 112;
13           APRNO      = 7;
14           WNDSZ      = 3;
15           REGSZ      = 21;  { NOPTR times WNDSZ }
16           NOPTR      = 7;
17           DUHLEN     = 10;
18           DUMMLEN    = 5;
19           NODATFLG   = 20;
20           IVCODE     = 193;
21           DVCODE     = 224;
22           ERRLEN     = 128;
23           GCDEST     = 2222;
24           GCSOUR     = 1111;
25           TPDUSZPL   = 1;
26           TPDUSZPC   = 192;
27           THPC       = 137;
28           GCTPDU     = 128;
29           GCTHR1     = 1;
30           GCTHR2     = 2;
31           GCTHR3     = 3;
32           GCTHR4     = 4;
33           THPL       = 12;
34           TRDLPL     = 8;
35           TRDLPC     = 136;
36           GCTRD1     = 7;
37           GCTRD2     = 8;
38           GCTRD3     = 9;
39           GCTRD4     = 10;
40           NREOT      = 99;
41           GCRESO     = 11;
42           UNDEF      = 99;
43
```

```
   44
   45   type                          { GLOBAL TYPES }
   46           std = record          { TPDUs }
   47                   fixp : packed array [1..FIXPLEN] of char;
   48                   oth  : packed array [1..OTHLEN] of char;
   49           end;
   50
   51           pstd = ^std;
   52
   53           contrl = array[0..7] of integer; { format for CRRG$ }
   54
   55           netXreq   = packed array[1..NETREQLEN] of char;
   56
   57           ddd = record
   58                   req : netXreq;
   59                   dum : array[1..DUMMLEN] of integer;
   60           end;
   61
   62           pdd = ^ddd;
   63
   64           tskk = record
   65                   nam : array [1..2] of integer;
   66           end;
   67
   68           rbuff = record { format for RCVD$ }
   69                   nam : tskk;
   70                   req : netXreq;
   71                   dum : array [1..DUMLEN] of integer;
   72           end;
   73
   74           buff = record  { format for SDAT$ }
   75                   req : netXreq;
   76                   dum : array [1..DUMLEN] of integer;
   77           end;
   78
   79           rddd = record
   80                   ok : array[1..2] of integer;
   81                   req : netXreq;
   82                   dum : array[1..DUMMLEN] of integer;
   83           end;
   84
   85           rpdd = ^rddd;
   86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rresid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  resid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmsg = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
  130
  131     .     errr = integer;
```

```
38
39
40  procedure werra(ptrwnd : pstd;
41                  rdcse : integer;
42                  mes : wmessXerr);
43
44  var
45        i   : integer;
46        j   : integer;
47
48  begin
49    writeln(tty, 'entering werra procedure');
50
51    case mes of      (* select the proper error status *)
52
53    five :
54      begin
55        writeln(tty, 'IMPOSSIBLE SITUATION');
56        writeln(tty, 'IMPOSSIBLE SITUATION')
57      end; (* five *)
58
59
60    six :
61      begin
62        writeln(tty, 'ILLEGAL USER REQUEST IN "CLOSED" STATE');
63        writeln(tty, 'ILLEGAL USER PRIMITIVE CODE IS', ord(ptrwnd^.fixp[1]))
64      end; (* six *)
65
66    seven :
67      begin
68        writeln(tty, 'ILLEGAL USER REQUEST IN "OPEN" STATE');
69        writeln(tty, 'ILLEGAL USER PRIMITIVE CODE IS', ord(ptrwnd^.fixp[1]))
70      end; (* seven *)
71
72    eight :
73      begin
74        writeln(tty, 'ILLEGAL USER REQUEST IN "WFCC" STATE');
75        writeln(tty, 'ILLEGAL USER PRIMITIVE CODE IS', ord(ptrwnd^.fixp[1]))
76      end; (* eight *)
77
78    nine :
79      begin
80        writeln(tty, 'ILLEGAL USER REQUEST IN "WFTRESP" STATE');
81        writeln(tty, 'ILLEGAL USER PRIMITIVE CODE IS', ord(ptrwnd^.fixp[1]))
```

```
 82       end; (* nine *)
 83
 84     ten  :
 85       begin
 86         writeln(tty, 'ILLEGAL USER REQUEST IN "WFNC" STATE');
 87         writeln(tty, 'ILLEGAL USER PRIMITIVE CODE IS', ord(ptrwnd^.fixp[1]))
 88       end; (* ten *)
 89
 90     eleven :
 91       begin
 92         writeln(tty, 'ER TPDU RECEIVED FROM TPRENTB');
 93         writeln(tty, 'CAUSE FOR REJECTION OF TPDU IS ', rjcse);
 94         writeln(tty, 'THE ERRONEOUS PACKET RETURNED BY TPRENTB IS');
 95         for j:= 0 to 7 do  (* assuming that ERRLEN = 128 *)
 96         begin
 97           for i :=((j * 8) + 1) to ((j + 1) * 16) do
 98             writeln(tty, ord(ptrwnd^.oth[10+i]));
 99           writeln(tty)
100         end;
101         writeln(tty, 'NETWORK DISCONNECTION REQUEST SENT TO NTRNSDA')
102       end; (* eleven *)
103
104     twelve :
105       begin
106         writeln(tty, 'INVALID "CC" TPDU RECEIVED FROM TPRENTB');
107         writeln(tty, 'CAUSE FOR REJECTION OF TPDU IS ', rjcse);
108         writeln(tty, 'NETWORK DISCONNECTION REQUEST SENT TO NTRNSDA')
109       end; (* twelve *)
110
111     thirteen :
112       begin
113         writeln(tty, 'ER TPDU RECEIVED FROM TPRENTB');
114         writeln(tty, 'CAUSE FOR REJECTION OF TPDU IS ', rjcse);
115         writeln(tty, 'THE ERRONEOUS PACKET RETURNED BY TPRENTB IS');
116         for j:= 0 to 7 do  (* assuming that ERRLEN = 128 *)
117         begin
118           for i := ((j * 8) + 1) to ((j + 1) * 16) do
119             writeln(tty, ord(ptrwnd^.oth[10+i]));
120           writeln(tty)
121         end;
122         writeln(tty, 'NETWORK DISCONNECTION REQUEST SENT TO NTRNSDA')
123       end; (* thirteen *)
124
```

```
125     fourteen :
126       begin
127         writeln(tty, 'ILLEGAL TPDU RECEIVED IN "CLOSED" STATE FROM TPRENTB');
128         writeln(tty, 'THE CODE OF THE TPDU IS ', ord(ptrwnd^.fixp[1]));
129         writeln(tty, 'TPDU IS IGNORED')
130       end; (* fourteen *)
131
132     fifteen :
133       begin
134         writeln(tty, 'ILLEGAL TPDU RECEIVED IN "WFNC" STATE FROM TPRENTB');
135         writeln(tty, 'THE CODE OF THE TPDU IS ', ord(ptrwnd^.fixp[1]));
136         writeln(tty, 'TPDU IS IGNORED')
137       end; (* fifteen *)
138
139     sixteen :
140       begin
141         writeln(tty, 'ILLEGAL TPDU RECEIVED FROM TPRENTB IN "WFTRESP" STATE');
142         writeln(tty, 'THE CODE OF THE TPDU IS ', ord(ptrwnd^.fixp[1]));
143         writeln(tty, 'TPDU IS IGNORED')
144       end; (* sixteen *)
145
146     seventeen :
147       begin
148         writeln(tty, 'ERRONEOUS DT TPDU RECEIVED FROM TPRENTB IN "OPEN" STATE');
149         writeln(tty, 'THE CODE OF THE TPDU IS ', ord(ptrwnd^.fixp[1]));
150         writeln(tty, 'ER TPDU IS SENT TO TPRENTB')
151       end; (* seventeen *)
152
153     eighteen :
154       begin
155         writeln(tty, 'ILLEGAL TPDU RECEIVED FROM TPRENTB IN "WFCC" STATE ');
156         writeln(tty, 'THE CODE OF THE TPDU IS ', ord(ptrwnd^.fixp[1]));
157         writeln(tty, 'ER TPDU IS SENT TO TPRENTB')
158       end; (* eighteen *)
159
160     nineteen :
161       begin
162         writeln(tty, 'INVALID "CR" TPDU RECEIVED FROM TPRENTB IN "CLOSED" STATE');
163         writeln(tty, 'CAUSE FOR REJECTION OF TPDU IS ', rjcse);
164         writeln(tty, '"DR" TPDU SENT TO TPRENTB')
165       end; (* nineteen *)
166
167     end;  (* case *)
```

```
  168     writeln(tty, 'leaving werra procedure')
  169   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      007504
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000530
COMPILATION TIME 47 SECONDS

PAS WERRA,WERRA/P42-WERRA
```

```
    1   (*$m-*)
    2   (*****************************************************************************
    3
    4      NAME:        WMESS
    5
    6      FUNCTION:    To write a NTRNSDA error message on the terminal in case
    7                   illegal requests are made by TPRENTA or NTRNSDB.
    8
    9      USAGE:       procedure wmess(messno : integer;
   10                                   stateXa : state;
   11                                   rreq : netXreq); extern;
   12
   13     INPUT PARAMETERS:    messno  - the type of message.
   14                          stateXa - the state of NTRNSDA.
   15                          rreq    - the inputed string.
   16
   17     OUTPUT PARAMETERS:   NONE
   18
   19     MODULES CALLED:      (The symbol '>' designates a primitive)
   20                          >halt
   21                          >writeln
   22
   23     EXIT CONDITIONS:
   24                          Normal exit conditions.
   25
   26     REMARKS:     NONE
   27
   28  *****************************************************************************)
   29
   30  {$I+NTRNSDA.DEF}
```

```
    1
    2   const
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           NETREQLEN  = 6;
    6           APRNO      = 7;
    7           WNDSZ      = 3;
    8           REGSZ      = 21;  { NOPTR times WNDSZ }
    9           NOPTR      = 7;
   10           DUMLEN     = 10;
   11           DUMMLEN    = 5;
   12           NODATFLG   = 21;
   13           TERMAFLAG  = 42;
   14           TERMBFLAG  = 40;
   15           TRNSDAFLAG = 43;
   16           TRNSDBFLAG = 41;
   17           CRCODE     = 224;
   18           CCCODE     = 208;
   19           DRCODE     = 128;
   20           DTCODE     = 240;
   21           ERCODE     = 112;
   22
   23   type
   24           std = record
   25                   fixp : packed array [1..FIXPLEN] of char;
   26                   oth  : packed array [1..OTHLEN] of char;
   27           end;
   28
   29           pstd = ^std;
   30
   31           contrl = array[0..7] of integer; { format for CRRG$ Executive Directive }
   32
   33           netXreq  = packed array[1..NETREQLEN] of char;
   34
   35           ddd = record
   36                   req : netXreq;
   37                   dum : array[1..DUMMLEN] of integer;
   38           end;
   39
   40           pdd = ^ddd;
   41
   42           tskk = record
   43                   num : array [1..2] of integer;
```

```
 44          end;
 45
 46          rbuff = record
 47                  nam : tskk;
 48                  req : netXreq;
 49                  dum : array [1..DUMLEN] of integer;
 50          end;
 51
 52          buff = record
 53                  req : netXreq;
 54                  dum : array [1..DUMLEN] of integer;
 55          end;
 56
 57          rddd = record
 58                  ok : array [1..2] of integer;
 59                  req : netXreq;
 60                  dum : array[1..DUMMLEN] of integer;
 61          end;
 62
 63          rpdd = ^rddd;
 64
 65          rwdbb = record
 66                  rapr : packed array[1..2] of char;
 67                  rvbadd : pstd;
 68                  rsiz : integer;
 69                  rregid : integer;
 70                  roffs : integer;
 71                  rlen : integer;
 72                  rwstats : integer;
 73                  rsrbuff : rpdd;
 74          end;
 75
 76          wdbb = record
 77                  apr : packed array[1..2] of char;
 78                  vbadd : pstd;
 79                  siz : integer;
 80                  regid : integer;
 81                  offs : integer;
 82                  len : integer;
 83                  wstats : integer;
 84                  srbuff : pdd;
 85          end;
 86
```

```
87          state     = (one, two, three, four);
88
89          idss = integer;
90
91          efnn = integer;
92
93          tmss = integer;
94
95          tntt = integer;
96
97          wndnoo = 1..NOPTR;
98
```

```
31
32
33  procedure wmess(messno : integer ;stateXa : state ; rreq : netXreq);
34
35  const                              { LOCAL VT100 TERMINAL CONSTANTS }
36        revideo = '[7m';
37        video   = '[0m';
38        up      = '[A';
39
40  begin
41
42     (* put in reverse video display mode *)
43     writeln(chr(27), revideo, chr(27), up);
44
45     if rreq = 'ustat ' then
46     begin
47       write(tty, 'NTRNSDA is in ');
48          case stateXa of
49
50            one : write(tty, 'IDLE ');
51            two : write(tty, 'OUT CONNECT PENDING ');
52            three: write(tty, 'IN CONNECT PENDING ');
53            four : write(tty, 'DATA TRANSFER READY ');
54          end;
55          writeln(tty, 'state.')
56     end
57     else case messno of
58
59        1 :  writeln (TTY,'tccon');
60        2 :  writeln (TTY,'tcind');
61        3 :  writeln (TTY,'tdind');
62        4 :  writeln (TTY,'tnodi');
63        5 :  writeln (TTY,'texdi');
64        6 :  begin
65          write(tty, 'NTRNSDA cannot accept a ', rreq, ' in ');
66          case stateXa of
67
68            one : write(tty, 'IDLE ');
69            two : write(tty, 'OUT CONNECT PENDING ');
70            three: write(tty, 'IN CONNECT PENDING ');
71            four : write(tty, 'DATA TRANSFER READY ');
72          end;
73          writeln(tty, 'state.')
74       end
```

```
   75
   76       end;       (* case *)
   77
   78       (* put back in normal mode *)
   79       writeln(chr(27), video, chr(27), up);
   80
   81   end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      002336
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000520
COMPILATION TIME 22 SECONDS

PAS WMESS,WMESS/P42=WMESS
```

```
1   (*$m-*)
2   (*****************************************************************************
3
4       NAME:       WQAB
5
6       FUNCTION:   To send information to task NTRNSDB via system directives
7                   send data (SDAT$) or send by reference (SREF$).
8
9       USAGE:      procedure wqab(ptrwnd : psld;
10                                 var wndno : wndnoo;
11                                 nXreq : netXreq;
12                                 id : integer;
13                                 bol : boolean); extern;
14                          .
15                          .
16                          .
17                  wqab(ptrwnd, wndno, nXreq, id, bol);
18
19      INPUT PARAMETERS:   ptrwnd  - the pointer to the buffer to be sent.
20                          wndno   - the window number to be used if bol = false.
21                          nXreq   - the string to be sent.
22                          id      - the region id.
23                          bol     - the boolean switch.
24
25      OUTPUT PARAMETERS:  wndno   - the next window number.
26
27      MODULES CALLED:     (The symbol '>' designates a primitive)
28
29                          >halt
30                          >writeln
31                          craw
32                          sref
33                          sdat
34
35      EXIT CONDITIONS:
36                          If there are any errors detected in calling CRAW$, SDAT$
37                          or SREF$, then an error message identifying the error and
38                          error code is printed onto standard I/O and the processor
39                          is then halted by using the 'halt' primitive.
40
41                          The return code of all these system directives, is set or
42                          greater or equal to 0 (zero) in case of successful completion
43                          and less than 0 in case of unsuccessful completion.
```

```
                        For more information on the use of these system routines
                        and their return codes, see the RSX-11M V3.2 Executive
                        Reference Manual.

    REMARKS:    NONE

*****************************************************************************)

{$I+TPRENTA.DEF}
```

```
    1
    2   const                          { GLOBAL CONSTANTS }
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           STRLEN     = 118;
    6           NETREQLEN  = 6;
    7           CRCODE     = 224;
    8           CCCODE     = 208;
    9           DTCODE     = 240;
   10           DRCODE     = 128;
   11           NDRCODE    = 120;
   12           ERCODE     = 112;
   13           APRNO      = 7;
   14           WNDSZ      = 3;
   15           REGSZ      = 21;  { NOPTR times WNDSZ }
   16           NOPTR      = 7;
   17           DUHLEN     = 10;
   18           DUMMLEN    = 5;
   19           NODATFLG   = 20;
   20           IVCODE     = 193;
   21           DVCODE     = 224;
   22           ERRLEN     = 128;
   23           GCDEST     = 2222;
   24           GCSOUR     = 1111;
   25           TPDUSZPL   = 1;
   26           TPDUSZPC   = 192;
   27           THPC       = 137;
   28           GCTPDU     = 128;
   29           GCTHR1     = 1;
   30           GCTHR2     = 2;
   31           GCTHR3     = 3;
   32           GCTHR4     = 4;
   33           THPL       = 12;
   34           TRDLPL     = 8;
   35           TRDLPC     = 136;
   36           GCTRD1     = 7;
   37           GCTRD2     = 8;
   38           GCTRD3     = 9;
   39           GCTRD4     = 10;
   40           NREOT      = 99;
   41           GCRESO     = 11;
   42           UNDEF      = 99;
   43
```

```
44
45   type                          { GLOBAL TYPES }
46        std = record             { TPDUs }
47                fixp : packed array [1..FIXPLEN] of char;
48                oth  : packed array [1..OTHLEN] of char;
49        end;
50
51        pstd = ^std;
52
53        contrl = array[0..7] of integer; { format for CRRG$ }
54
55        netXreq   = packed array[1..NETREQLEN] of char;
56
57        ddd = record
58                req : netXreq;
59                dum : array[1..DUMMLEN] of integer;
60        end;
61
62        pdd = ^ddd;
63
64        tskk = record
65                nam : array [1..2] of integer;
66        end;
67
68        rbuff = record { format for RCVD$ }
69                nam : tskk;
70                req : netXreq;
71                dum : array [1..DUMLEN] of integer;
72        end;
73
74        buff = record  { format for SDAT$ }
75                req : netXreq;
76                dum : array [1..DUMLEN] of integer;
77        end;
78
79        rddd = record
80                ok : array[1..2] of integer;
81                req : netXreq;
82                dum : array[1..DUMMLEN] of integer;
83        end;
84
85        rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rregid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  regid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmss = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
  130
  131           errr = integer;
```

```
54
55
56  procedure buffclear(efn : efnn); extern;
57
58  procedure buffso(efn : efnn); extern;
59
60  procedure buffwait(efn : efnn); extern;
61
62  procedure wqab(ptrwnd : pstd;
63                 var wndno : wndnoo;
64                 nXreq : netXreq;
65                 id : integer;
66                 bol : boolean);
67
68  var
69          bf      : buff;
70          efn     : efnn;
71          ids     : idss;
72          isrb    : ddd;
73          swdb    : wdbb;
74          tsk     : tskk;
75          wdb     : wdbb;
76
77  procedure craw(var wdb : wdbb; var ids : idss); extern(fortran);
78
79  procedure send(var tsk : tskk;
80                 var bf  : buff;
81                 var efn : efnn;
82                 var ids : idss); extern(fortran);
83
84  procedure sref(var tsk: tskk;
85                 var efn : efnn;
86                 var swdb : wdbb;
87                 var isrb : ddd;
88                 var ids : idss); extern(fortran);
89  begin
90
91  { writeln(tty, ' entering wqab procedure');}
92
93    (* set up task name = 'TRNSDB' *)
94    tsk.nam[1] := 32734;
95    tsk.nam[2] := 30562;
96
97    if bol then  (* send data *)
```

```
 98     begin
 99       (* put string in buffer to be sent *)
100       bf.req := nXreq;
101
102       (* send the data to the TRNSDB task *)
103       send(tsk, bf, efn, ids);
104       if ids < 0 then
105       begin
106          writeln(tty, 'error in calling SEND in SENDD,  error code = ', ids);
107          halt
108       end
109     end
110     else
111     begin  (* send by reference to task TRNSDB *)
112
113       (* create a send window in the region *)
114       wdb.apr[2] := chr(APRNO);   (* Active page register 7 *)
115       wdb.siz := WNDSZ;           (* the size of the window in 32-word blocks *)
116       wdb.regid := id;            (* region id *)
117                                   (* or 0 for task region *)
118                                   (* specified only if WS.MAP is set *)
119       wdb.offs := (wndno-1)*WNDSZ;(* offset in region (32W blocks) at which
120                                      the window is to start mapping *)
121                                   (* specified only if WS.MAP is set *)
122       wdb.len := WNDSZ;           (* length to map (32W blocks) *)
123                                   (* or 0 if the length is to default to either
124                                      the size of the window or the space remaining
125                                      in the region *)
126                                   (* specified only if WS.MAP is set *)
127       wdb.wstats := 386;          (* window status word *)
128                                   (* bits set:
129                                            WS.64B = 256
130                                            WS.MAP = 128
131                                            WS.WRT = 2                *)
132       craw(wdb, ids);
133   {   writeln(tty, 'window created ids =', ids);
134       writeln(tty, 'Id assigned to the window = ',ord(wdb.apr[1]));
135       writeln(tty, 'virtual base address = ', ord(wdb.vbadd));
136       writeln(tty, 'length actually mapped in 32-word blocks = ', wdb.len);
137       writeln(tty, 'window status word = ', wdb.wstats);}
138       if ids < 0 then
139       begin
140          writeln(tty, ' error in using craw,  error code = ',ids);
```

```
  141          halt
  142       end;
  143
  144       (* stick buffer into the send window *)
  145       wdb.vbadd^.fixp := ptrwnd^.fixp;
  146       wdb.vbadd^.oth := ptrwnd^.oth;
  147
  148       (* prepare window definition block for send by reference *)
  149       swdb.regid := id;              (* id of the region to be sent by reference *)
  150       swdb.offs := (wndno-1) * WNDSZ;(* offset in region (32W blocks) *)
  151       swdb.len := WNDSZ;             (* length to map (32W blocks) *)
  152       swdb.wstats := 3;              (* window status word *)
  153                                      (* bits set:
  154                                                 WS.WRT = 2
  155                                                 WS.RED = 1          *)
  156
  157       (* put strings into the control send by reference buffer *)
  158       isrb.rea := nXrea;
  159
  160       sref(tsk, efn, swdb, isrb, ids);
  161    {  writeln(tty, 'ids for sref is = ', ids);}
  162       if ids < 0 then
  163       begin
  164          writeln(tty, ' error in using sref,  error code = ',ids);
  165          halt
  166       end;
  167
  168       (* fix up window number to point to the next window in the circular queue *)
  169       if wndno = NOPTR then
  170         wndno := 1
  171       else
  172         wndno := wndno + 1;
  173
  174     end
  175
  176    {writeln(tty,' leaving wqab procedure')}
  177    end.
```

```
NO ERROR DETECTED
TOTAL PROGRAM SIZE      001744
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000642
COMPILATION TIME 36 SECONDS
```

```
1   (*$m-*)
2   (*****************************************************************************)
3
4       NAME:           WTERMA
5
6       FUNCTION:       Write an appropriate legal message on the terminal A
7                       which could be TCIND, TDIND, TNODI or TCCON with
8                       appropriate parameter values for the above primitives,
9                       as per ISO Transport Protocol Specs. Apr. 1983.
10
11      USAGE:          procedure wterma(ptrwnd : pstd;
12                                      mes : wmessXsts); extern;
13                                .
14                                .
15                                .
16                      wterma(ptrwnd, mes);
17
18      INPUT PARAMETERS:       ptrwnd  - the pointer to the TPDU received.
19                              mes     - the message number.
20
21      OUTPUT PARAMETERS:      NONE
22
23      MODULES CALLED:         (The symbol '>' designates a primitive)
24
25                              >char
26                              >ord
27                              >writeln
28                              val
29
30      EXIT CONDITIONS:
31                              Normal exit conditions.
32
33      REMARKS:
34                      A lot of the code is for generating a nice display on a
35                      VT100 compatible terminal.
36
37                      Most of the fields in a TPDU are outputed using the 'ord'
38                      primitive.  This is because the fields are stored in byte
39                      (character) format.
40
41                      The VAL function is used to associate two bytes together and
42                      form an integer.
43
```

```
   44   ********************************************************************************)
   45
   46   {$I+TPRENTA.DEF}
```

```
    1
    2   const                       { GLOBAL CONSTANTS }
    3           FIXPLEN    = 8;
    4           OTHLEN     = 130;
    5           STRLEN     = 118;
    6           NETREQLEN  = 6;
    7           CRCODE     = 224;
    8           CCCODE     = 208;
    9           DTCODE     = 240;
   10           DRCODE     = 128;
   11           NDRCODE    = 120;
   12           ERCODE     = 112;
   13           APRNO      = 7;
   14           WNDSZ      = 3;
   15           REQSZ      = 21;   { NOPTR times WNDSZ }
   16           NOPTR      = 7;
   17           DUMLEN     = 10;
   18           DUMMLEN    = 5;
   19           NODATFLG   = 20;
   20           IVCODE     = 193;
   21           DVCODE     = 224;
   22           ERRLEN     = 128;
   23           GCDEST     = 2222;
   24           GCSOUR     = 1111;
   25           TPDUSZPL   = 1;
   26           TPDUSZPC   = 192;
   27           THPC       = 137;
   28           GCTPDU     = 128;
   29           GCTHR1     = 1;
   30           GCTHR2     = 2;
   31           GCTHR3     = 3;
   32           GCTHR4     = 4;
   33           THPL       = 12;
   34           TRDLPL     = 8;
   35           TRDLPC     = 136;
   36           GCTRD1     = 7;
   37           GCTRD2     = 8;
   38           GCTRD3     = 9;
   39           GCTRD4     = 10;
   40           NREOT      = 99;
   41           GCRESO     = 11;
   42           UNDEF      = 99;
   43
```

```
44
45   type                          { GLOBAL TYPES }
46           std = record          { TPDUs }
47                   fixp : packed array [1..FIXPLEN] of char;
48                   oth  : packed array [1..OTHLEN] of char;
49           end;
50
51           pstd = ^std;
52
53           contrl = array[0..7] of integer; { format for CRRG$ }
54
55           netXreq   = packed array[1..NETREQLEN] of char;
56
57           ddd = record
58                   req : netXreq;
59                   dum : array[1..DUMMLEN] of integer;
60           end;
61
62           pdd = ^ddd;
63
64           tskk = record
65                   nam : array [1..2] of integer;
66           end;
67
68           rbuff = record { format for RCVD$ }
69                   nam : tskk;
70                   req : netXreq;
71                   dum : array [1..DUMLEN] of integer;
72           end;
73
74           buff = record  { format for SDAT$ }
75                   req : netXreq;
76                   dum : array [1..DUMLEN] of integer;
77           end;
78
79           rddd = record
80                   ok : array[1..2] of integer;
81                   req : netXreq;
82                   dum : array[1..DUMMLEN] of integer;
83           end;
84
85           rpdd = ^rddd;
86
```

```
 87          rwdbb = record { format for CRAW$ and RREF$ }
 88                  rarr : packed array[1..2] of char;
 89                  rvbadd : pstd;
 90                  rsiz : integer;
 91                  rregid : integer;
 92                  roffs : integer;
 93                  rlen : integer;
 94                  rwstats : integer;
 95                  rsrbuff : rpdd;
 96          end;
 97
 98          wdbb = record  { format for CRAW$ and SREF$ }
 99                  arr : packed array[1..2] of char;
100                  vbadd : pstd;
101                  siz : integer;
102                  regid : integer;
103                  offs : integer;
104                  len : integer;
105                  wstats : integer;
106                  srbuff : pdd;
107          end;
108
109          state     = (CLOSED, OPEN, WFCC, WFTRESP, WFNC); { states of TPRENTA }
110
111          opXsts    = (opens, closeds, inprogress);
112
113          wmessXsts = (zero,one,two,three,four);
114
115          wmessXerr = (five,six,seven,eight,nine,ten,eleven,
116          twelve,thirteen,fourteen,fifteen,sixteen,seventeen,eighteen,
117          nineteen,twenty,twentyone);
118
119          idss = integer;
120
121          efnn = integer;
122
123          tmss = integer;
124
125          tntt = integer;
126
127          wndnoo = 1..NOPTR;
128
129          arr = packed array[1..2] of char;
```

```
  130
  131         errr := integer;
```

```
47
48
49  procedure wterma(ptrwnd : pstd;
50                   mes : wmessXsts);
51
52
53  function val(c1 : char;
54               c2 : char) : integer; extern;
55
56  begin                                    (* wterma *)
57
58  { writeln(tty, 'entering wterma procedure');}
59
60    (* put in reverse video *)
61    writeln(tty, chr(27), '[7m', chr(27), '[10;24r', chr(27), '[10;1f', chr(27), '[A');
62
63    case mes of                            (* select the appropriate action *)
64
65    zero:
66      begin
67        writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
68        writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
69        writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
70        writeln(tty, '<< TRANSPORT DISCONNECT INDICATION >>');
71        writeln(tty);
72        writeln(tty, 'the TRANSPORT CONNECTION is now disconnected :');
73        writeln(tty); writeln(tty); writeln(tty); writeln(tty);
74        writeln(tty); writeln(tty); writeln(tty);
75        writeln(tty); writeln
76      end;                                 (* zero *)
77
78    one :
79      begin
80        writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
81        writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
82        writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
83        writeln(tty, '<< TRANSPORT CONNECT INDICATION >>');
84        writeln(tty, 'SOURCE ADDRESS : ',
85                val(ptrwnd^.fixp[5], ptrwnd^.fixp[6]));
86        writeln(tty, 'DESTINATION ADDRESS : ',
87                val(ptrwnd^.fixp[3], ptrwnd^.fixp[4]));
88
89        writeln(tty, 'quality of service parameters are :');
90        writeln(tty, 'CALLING TSAP-ID : ', ord(ptrwnd^.oth[4]));
```

```
 91          writeln(tty, 'CALLED TSAP-ID  : ', ord(ptrwnd^.oth[5]));
 92          writeln(tty, 'TPDU SIZE       : ', ord(ptrwnd^.oth[10]));
 93          writeln(tty, 'THROUGHPUT      : ',
 94                  val(ptrwnd^.oth[11], ptrwnd^.oth[12]),
 95                  val(ptrwnd^.oth[13], ptrwnd^.oth[14]),
 96                  val(ptrwnd^.oth[15], ptrwnd^.oth[16]),
 97                  val(ptrwnd^.oth[17], ptrwnd^.oth[18]));
 98          writeln(tty, 'TRANSIT DELAY   : ',
 99                  val(ptrwnd^.oth[27], ptrwnd^.oth[28]),
100                  val(ptrwnd^.oth[29], ptrwnd^.oth[30]),
101                  val(ptrwnd^.oth[31], ptrwnd^.oth[32]),
102                  val(ptrwnd^.oth[33], ptrwnd^.oth[34]));
103          writeln(tty);
104          writeln(tty, 'do you accept this TRANSPORT CONNECTION  ???');
105          writeln(tty, 'if you do, please type in a TCRES with appropriate parameters');
106          writeln(tty, 'OR type in a TDREQ with appropriate parameters if you ');
107          writeln(tty, 'wish to disconnect')
108        end;                                    (* one *)
109
110      two:
111        begin
112          writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
113          writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
114          writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
115          writeln(tty, '<< TRANSPORT CONNECT CONFIRMATION >>');
116          writeln(tty, 'RESPONDING ADDRESS : ',
117                  val(ptrwnd^.fixp[5], ptrwnd^.fixp[6]));
118
119     (* destination address not required as per primitive parameter definition *)
120     (* page.12. of ISO Transport Protocol Specs. Apr. 1983 *)
121
122          writeln(tty);
123          writeln(tty, 'quality of service parameters are :');
124          writeln(tty, 'CALLING TSAP-ID : ', ord(ptrwnd^.oth[4]));
125          writeln(tty, 'CALLED TSAP-ID  : ', ord(ptrwnd^.oth[5]));
126          writeln(tty, 'TPDU SIZE       : ', ord(ptrwnd^.oth[10]));
127          writeln(tty, 'THROUGHPUT      : ',
128                  val(ptrwnd^.oth[11], ptrwnd^.oth[12]),
129                  val(ptrwnd^.oth[13], ptrwnd^.oth[14]),
130                  val(ptrwnd^.oth[15], ptrwnd^.oth[16]),
131                  val(ptrwnd^.oth[17], ptrwnd^.oth[18]));
132          writeln(tty, 'TRANSIT DELAY   : ',
133                  val(ptrwnd^.oth[27], ptrwnd^.oth[28]),
```

```
134                 val(ptrwnd^.oth[29], ptrwnd^.oth[30]);
135                 val(ptrwnd^.oth[31], ptrwnd^.oth[32]);
136                 val(ptrwnd^.oth[33], ptrwnd^.oth[34]));
137           writeln(tty);
138           writeln(tty, 'TRANSPORT CONNECTION is now OPEN');
139           writeln(tty, 'please type in user data using the TNODR primitive with');
140           writeln(tty, 'appropriate parameters OR type in a TDREQ with appropriate');
141           writeln(tty, 'parameters if you wish to disconnect')
142         end;                                    (* two *)
143
144       three:
145         begin
146           writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
147           writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
148           writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
149           writeln(tty, '<< TRANSPORT DATA INDICATION >>');
150           writeln(tty);
151           writeln(tty, 'the data sent in by the responder is :');
152           writeln(tty);
153           writeln(tty, chr(27), '[5m', chr(27), '[A');
154           write(tty,
155                   ptrwnd^.oth[3..(ord(ptrwnd^.oth[1]) + 2)]);
156           writeln(tty, chr(27), '[0m', chr(27), '[7m', chr(27), '[A');
157           writeln(tty);
158           writeln(tty, 'please type in user data in TNODR primitive with');
159           writeln(tty, 'appropriate parameters');
160           writeln(tty, 'OR type in a TDREQ with appropriate parameters if you ');
161           writeln(tty, 'wish to disconnect');
162           writeln(tty); writeln(tty); writeln(tty)
163         end;                                    (* three *)
164
165       four:
166         begin
167           writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
168           writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
169           writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);writeln(tty);
170           writeln(tty, '<< TRANSPORT DISCONNECT INDICATION >>');
171           writeln(tty);
172           writeln(tty, 'the TRANSPORT CONNECTION is now disconnected :');
173           writeln(tty);
174           writeln(tty, 'the reason for disconnection is :');
175           writeln(tty, ord(ptrwnd^.fixp[8]));
176           writeln(tty, 'additional information for disconnection is :');
```

```
  177         writeln(tty);
  178         writeln(tty, chr(27), '[5m', chr(27), '[A');
  179         write(tty,
  180                   ptrwnd^.oth[3..(ord(ptrwnd^.oth[1]) + 2)]);
  181         writeln(tty, chr(27), '[0m', chr(27), '[7m', chr(27), '[A');
  182         writeln(tty);writeln(tty);writeln(tty);writeln(tty)
  183       end                                   (* four *)
  184     end;                                    (* case *)
  185
  186     (* put back in normal video *)
  187     writeln(tty, chr(27), '[0m', chr(27), '[7;24r', chr(27), '[7;1f', chr(27), '[A');
  188
  189   { writeln(tty, 'leaving wterma procedure')}
  190   end.

NO ERROR DETECTED
TOTAL PROGRAM SIZE      015026
OUTERMOST DATA SIZE     000002
RESERVED STACK & HEAP   000520
COMPILATION TIME 131 SECONDS

PAS WTERMA,WTERMA/P42-WTERMA
```