IMPLEMENTAÇÃO DE UM SISTEMA DE COMUNICAÇÕES PARA O SISTEMA OPERACIONAL PLURIX

Sílvia Maria Barbosa Figueira

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Newton Faller, Ph.D.
(presidente)

Prof. Edil S. Tavares Fernandes, Ph.D.

Prof. Pedro Salenbauch, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 1991

FIGUEIRA, SÍLVIA MARIA BARBOSA

Implementação de um Sistema de Comunicações para o Sistema Operacional PLURIX [Rio de Janeiro] 1991

V, 90 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1991)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas Operacionais

I. COPPE/UFRJ II. Título (série).

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

IMPLEMENTAÇÃO DE UM SISTEMA DE COMUNICAÇÕES
PARA O SISTEMA OPERACIONAL PLURIX

Sílvia Maria Barbosa Figueira

Abril, 1991

Orientador: Prof. Newton Faller, Ph.D.
Programa: Engenharia de Sistemas e Computação

Este trabalho consiste na definição e implementação de um sistema de comunicações para o Sistema Operacional PLURIX. Este sistema de comunicações foi desenvolvido baseado na família de protocolos TCP/IP e no padrão para interface de transporte estabelecido pelo X/OPEN.

Apresentamos, inicialmente, a família de protocolos TCP/IP e as definições de interface de transporte mais utilizadas. Mostramos, então, os aspectos mais importantes e significativos tanto da definição, quanto da implementação do sistema de comunicações. E, para finalizar, analisamos o trabalho realizado, apresentando suas futuras extensões.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

THE IMPLEMENTATION OF A COMMUNICATION SYSTEM
FOR THE PLURIX OPERATING SYSTEM

Sílvia Maria Barbosa Figueira

April, 1991

Thesis Supervisor: Prof. Newton Faller, Ph.D.
Department: Systems Engineering and Computer Science

This work consists of the definition and the implementation of a communication system for the PLURIX Operating System. This communication system was developed based on the TCP/IP protocols family and on the transport interface standard defined by X/OPEN.

We present, inicially, the TCP/IP protocols family and the most used transport interface definitions. Then we show the main aspects of the definition and of the implementation of the communication system. Finally, we analyze the work done, and present its future extensions.

ÍNDICE

CAPÍTULO I

INTRODUÇÃO

I.1. Apresentação

Nas duas primeiras décadas da existência dos computadores, sua utilização era bastante restrita e totalmente centralizada. Assim, as necessidades computacionais de uma organização eram, em geral, supridas por poucos computadores de grande porte, funcionando isoladamente. Atualmente, este modelo foi substituído por outro, no qual as necessidades computacionais de uma organização são supridas por sistemas formados por um grande número de computadores de portes variados, que embora sejam autônomos, se encontram interligados. Estes sistemas são conhecidos por redes de computadores. Responsáveis pela intercomunicação entre sistemas computacionais, homogêneos ou não, as redes fazem, atualmente, parte do dia a dia de um grande número de usuários de computadores. Cada vez mais versáteis, elas tornam possível o compartilhamento de recursos (informações, serviços e periféricos) entre computadores, contribuindo para um aumento de suas potencialidades.

O sistema de comunicações é a parte do sistema computacional responsável por tornar disponível aos

usuários os serviços oferecidos pelas redes. Ele é responsável pela manutenção de toda a comunicação envolvida com o sistema. Como a possibilidade de integração dos mais diversos ambientes computacionais vem gerando uma interdependência entre os diversos sistemas, o intercâmbio de recursos vem se tornando imprescindível aos usuários, transformando, assim, os sistema de comunicações em mecanismos inerentes aos sistemas computacionais.

O PLURIX (*) [1-2-3-4] é um sistema operacional com filosofia UNIX (**) [5] definido e implementado no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro (NCE/UFRJ). Por ser o PLURIX utilizado como base para o desenvolvimento de novos projetos na área de sistemas operacionais - fato que se deve ao total controle que se tem sobre ele, já que foi integralmente desenvolvido pela equipe do NCE -, era de suma importância dotá-lo de um sistema de comunicações, sem o qual ele não teria acesso às redes e às facilidades por elas oferecidas.

Assim, visando dotar o PLURIX de um mecanismo capaz de tornar disponíveis aos seus usuários os serviços provenientes das redes de comunicações, foi definido e implementado um sistema de comunicações baseado nos protocolos TCP/IP [6].

(*) PLURIX é Marca Registrada do NCE/UFRJ.
(**) UNIX é Marca Registrada da AT&T.

Este trabalho se baseia na experiência adquirida ao longo da definição e da implementação do sistema de comunicações do PLURIX. Nele são discutidas as decisões tomadas na fase de definição, a estrutura básica do sistema como um todo e as soluções empregadas para contornar as incompatibilidades com o sistema operacional.

Uma grande motivação à execução desta dissertação foi o fato de haver pouco material relativo às questões de implementação de sistemas de comunicações na literatura. Este problema é abordado por WATSON e MAMRAK [7] que, inclusive, incentivam a publicação de experiências com a implementação de protocolos. A maior parte dos artigos existentes, porém, se referem à implementação de sistemas operacionais voltados para a comunicação, como é o caso do x-Kernel [8], do Accent [9] e do V-Kernel [10], e não para a ambientação de protocolos em um sistema operacional já existente, como é o caso do PLURIX. Uma exceção é a descrição feita por COMER [11] acerca da implementação do sistema de comunicações do Sistema Operacional XINU. No entanto, esta descrição é bastante simplificada, não abordando, por exemplo, o protocolo TCP.

O sistema de comunicações implementado no PLURIX permite que os programas dos seus usuários se comuniquem com programas de usuários de outros sistemas computacionais, desde que estes sistemas utilizem a família de protocolos TCP/IP. No seu estágio atual, enquanto a interface para Ethernet [12] não é concluída,

exige-se ainda que estes sistemas possuam uma interface serial utilizando o protocolo SLIP [13].

I.2. Organização

Este trabalho contém, além deste de introdução, mais cinco capítulos. O capítulo II faz uma abordagem geral sobre o modelo "internet" e sobre a família de protocolos TCP/IP, na qual é baseada a implementação do sistema de comunicações do PLURIX. Um enfoque especial é dado aos protocolos TCP, UDP e IP, que formam o conjunto básico necessário para a viabilização da comunicação.

No capítulo III, são apresentados os padrões de interface de transporte mais utilizados em sistemas operacionais com filosofia UNIX. É mostrada, ainda, a funcionalidade exigida de uma interface de transporte.

O capítulo IV descreve a implementação do sistema de comunicações do PLURIX. Inicialmente, é apresentada a interface de transporte escolhida e a forma utilizada para a sua instalação. Em uma outra seção, é feita a apresentação da implementação do sistema de comunicações propriamente dito. Na última seção mostramos as estruturas e os mecanismos básicos utilizados.

No capítulo V, apresentamos os testes realizados ao longo da implementação, bem como a maneira encontrada para

a utilização imediata do sistema.

Finalmente, no capítulo VI, são mostradas as conclusões e as extensões pretendidas para o sistema. Fazemos, assim, uma avaliação da escolha da família de protocolos e da interface de transporte. Analisamos, ainda, as estruturas e algoritmos utilizados em alguns pontos críticos do sistema.

CAPÍTULO II

A FAMÍLIA DE PROTOCOLOS TCP/IP

A utilização de redes locais de comunicação faz, cada vez mais, parte do dia a dia de um grande número de usuários de computadores. Estes usuários começaram a sentir, há algum tempo, a necessidade de um ambiente global, no qual usuários das mais diversas redes pudessem se comunicar. No entanto, as muitas redes existentes atualmente são ainda restritas, dando a impressão de fragmentação na comunicação entre computadores. A intenção de se obter uma ligação global, esbarra no fato de que nenhuma tecnologia disponível atende às necessidades de todos os tipos de usuários. Em outras palavras, enquanto as redes locais possuem altas taxas de transmissão, mas são geograficamente limitadas, as redes de longa distância têm grande alcance, mas suas taxas de transmissão deixam muito a desejar. Assim, resta a alternativa de continuar utilizando as mesmas tecnologias existentes, e solucionar o problema da fragmentação através da interconexão de redes (homogêneas ou heterogêneas), fazendo, assim, com que o usuário tenha a ilusão de estar utilizando uma rede única de grande amplitude. A esta forma de rede tem-se dado o nome de "internetwork" ou "internet" [6].

A "internet" é composta por diversas redes

interligadas por computadores denominados "gateways". Cada "gateway" faz a conexão entre duas ou mais redes, devendo, portanto, conhecer a tecnologia utilizada em cada uma delas. Desta forma, os "gateways" são os responsáveis pelo percurso das mensagens, que passam de rede em rede (através dos "gateways") até chegar ao computador destino, na sua respectiva rede.

O sistema de endereçamento dos computadores pertencentes à "internet" é muito importante e seu controle deve ser centralizado para que não haja problemas de conflito resultante da duplicação de endereços. Assim, a cada computador é associado um endereço de 32 bits que é usado em todas as comunicações com aquele computador. Este endereço é formado pelo seu endereço na sua respectiva rede e pelo endereço desta rede na "internet". Quando uma mensagem é enviada por um computador, seu destino, identificado pelo endereço nela contido, pode estar na própria rede, sendo entregue diretamente, ou em outra, devendo ser enviada a um "gateway". Como uma rede pode estar ligada a mais de uma rede, possuindo, assim, mais de um "gateway", os computadores, devem ser capazes de identificar qual destes "gateways" inicia a melhor rota para a mensagem. No "gateway" (caso ele mesmo não seja o destino), a mesma situação ocorre, isto é, se o destino da mensagem está em uma rede ao qual o "gateway" está ligado, a mensagem é enviada diretamente, caso contrário, a mensagem é encaminhada a outro "gateway", e assim sucessivamente.

Para que os diversos computadores envolvidos em uma rede possam interagir, é preciso que seja definida uma linguagem comum que todos entendam. Esta linguagem comum é implementada através dos chamados protocolos de comunicação. Protocolos se constituem, assim, em conjuntos de regras de comunicação respeitadas por todos os computadores que façam parte da rede.

A fim de tornar a topologia da "internet" transparente ao usuário, e com isso contribuir para uma maior portabilidade dos programas de aplicação, é utilizado um protocolo, cuja execução é responsável pela manipulação do percurso das mensagens ao longo da "internet". Por outro lado, como a comunicação entre dois computadores ocorre, na realidade, entre dois programas de aplicação em execução nestes computadores, quando uma mensagem chega ao computador destino ela deve ainda ser encaminhada a um usuário. Por esta razão, é necessário que haja um protocolo, cuja execução seja capaz de garantir o recebimento de uma mensagem pelo usuário correto.

A utilização destes protocolos na comunicação permite que os usuários desenvolvam programas de aplicação totalmente independentes do computador e da rede utilizados. Os sistemas de comunicação complexos utilizam, para manipular todas as tarefas relativas à transmissão, um conjunto de protocolos, que chamamos de família de protocolos. Os vários protocolos de uma família são organizados em níveis e cada nível é responsável por uma

parte específica das tarefas.

Enviar uma mensagem de um programa de aplicação em um computador para um programa de aplicação em outro computador, significa que a mensagem passará por sucessivos níveis de protocolos no computador origem, será enviada pela rede e, no computador destino, passará pelos mesmos protocolos pelos quais havia passado no computador origem, só que na ordem inversa, como mostra a figura II.1. Conceitualmente, a organização dos protocolos em níveis é feita de forma que o nível "n" no destino recebe exatamente a mesma mensagem enviada pelo nível "n" na origem. Assim, em cada nível do computador origem a mensagem é acrescida de informações que serão utilizadas e retiradas da mensagem pelo nível correspondente no computador destino.

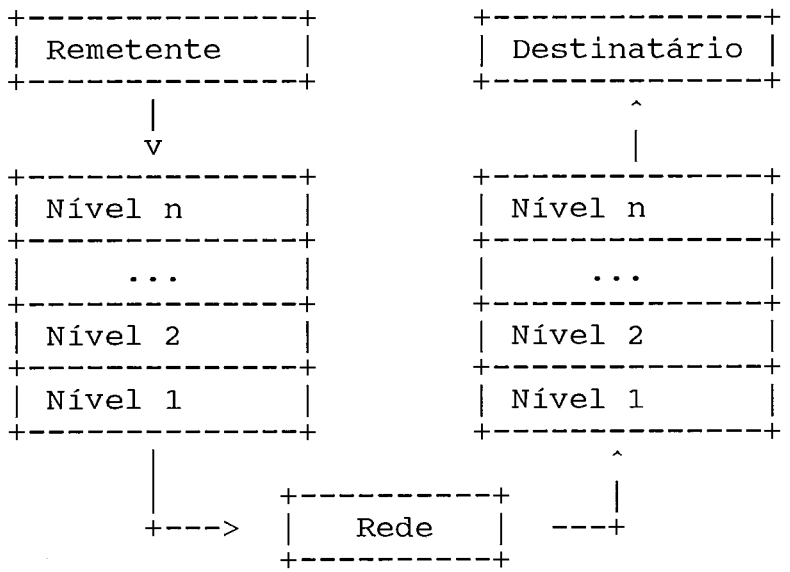

Figura II.1

Organização conceitual dos protocolos em níveis.

Em cada nível, um protocolo toma decisões sobre a correção de mensagens e escolhe uma ação apropriada baseada no tipo da mensagem e no destino da mesma. Por exemplo, quando um computador recebe uma mensagem, o protocolo de um determinado nível deve decidir se aceita a mensagem ou se a envia para outro computador. Em outro nível, o protocolo deve decidir qual programa de aplicação deve receber a mensagem.

Dois modelos se destacam na organização dos protocolos em níveis. O primeiro, baseado em um trabalho realizado pela ISO (International Standards Organization), é conhecido por "ISO Reference Model of Open System Interconnection (OSI)", ou simplesmente por modelo ISO [14]. Este modelo contém sete níveis conceituais

organizados como mostra a figura II.2. Eles são os seguintes:

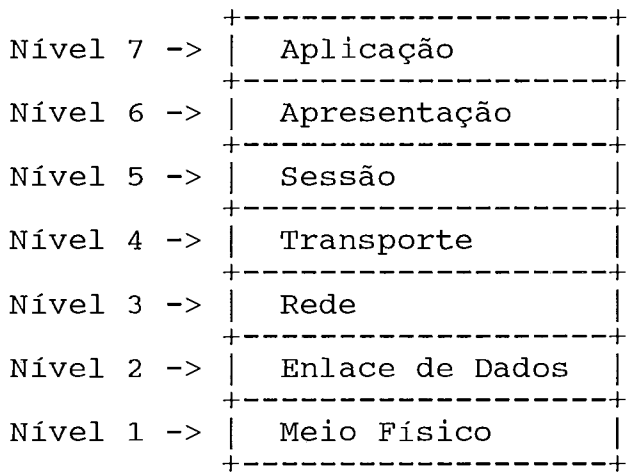

Figura II.2

Níveis conceituais do modelo ISO.

Aplicação

Ou nível 7, contém os protocolos utilizados pelos programas de aplicação que são os responsáveis por garantir aos usuários o acesso à rede. Entre estes protocolos se encontram, por exemplo, as especificações para a implementação de correio eletrônico.

Apresentação

Ou nível 6, é responsável pela representação da informação utilizada no nível 7, possuindo, por exemplo, rotinas para a compressão de dados ou para a conversão de imagens gráficas em cadeias de bits para a transmissão pela rede.

Sessão

Ou nível 5, é responsável pela organização e sincronização do diálogo entre entidades do nível de apresentação em comunicação. São fornecidos, assim, serviços para o estabelecimento de uma conexão de sessão entre duas entidades de apresentação, através do uso de uma conexão de transporte.

Transporte

Ou nível 4, provê serviços transparentes de transmissão para os dados do nível 5, sendo responsável pelas questões de confiabilidade e eficiência. É também responsável pela característica fim-a-fim da transferência de dados, ou seja, pela formação da conexão entre usuários.

Rede

Ou nível 3, fornece uma trajetória de conexão entre um par de entidades do nível de transporte, possivelmente passando por computadores intermediários. Este nível é o responsável pela operação da rede, possuindo os algoritmos de roteamento e de controle de congestionamento.

Enlace de Dados

Ou nível 2, é responsável por conhecer as características do meio físico a ser utilizado na comunicação, funcionando como uma interface entre ele e o sistema. Neste nível são detectados e corrigidos

os possíveis erros ocorridos na transferência de dados.

Meio Físico

Ou nível 1, é a rede propriamente dita, onde ocorre a transmissão efetiva dos dados.

O segundo modelo de organização de protocolos não surgiu de um comitê de padronização, e sim do trabalho que levou ao desenvolvimento da família de protocolos TCP/IP [6], criada para permitir que computadores localizados ao longo de uma "internet" compartilhassem recursos e serviços. O nome TCP/IP se deve ao fato de serem o TCP e o IP os protocolos mais conhecidos da família "Internet" que, na realidade, é formada por muitos outros protocolos.

Por ter sido desenvolvido utilizando o Sistema Operacional UNIX da Universidade da Califórnia em Berkeley, EUA, este conjunto de protocolos tem se tornado um padrão de fato para os sistemas operacionais com filosofia UNIX, sendo, por isso, escolhido para ser utilizado no sistema de comunicações do PLURIX. Outro fator que favoreceu a escolha dos protocolos TCP/IP, em detrimento dos protocolos definidos pelo padrão ISO, foi a sua utilização em um número maior de sistemas. Embora haja uma tendência geral em direção à utilização do padrão ISO, esta mudança ocorrerá a longo prazo, justificando ainda a implementação dos protocolos TCP/IP.

Os protocolos TCP/IP foram distribuídos em quatro níveis conceituais posicionados acima da rede local (nível 1, ou meio físico, do modelo ISO), que, neste modelo, não constitui um nível específico. As seções seguintes fazem uma descrição destes níveis, cuja organização é mostrada na figura II.3.

Figura II.3

Organização da família TCP/IP sobre a Rede Local.

II.1. Nível de Aplicação

Neste nível, correspondente aos níveis 5, 6 e 7 do modelo ISO, estão os protocolos utilizados pelos programas de aplicação. É através da execução destes programas que os usuários têm acesso à rede. Cada programa executa uma tarefa específica através da comunicação, realizada com a utilização de um protocolo próprio, com um outro programa de aplicação (normalmente em outro computador). Entre estes protocolos podemos citar o FTP (File Transfer Protocol) [15] que é utilizado na transferência de arquivos e o TELNET (Network Terminal Protocol) [16] que é

utilizado na implementação de um terminal virtual.

II.2. Nível de Transporte

Neste nível, correspondente ao nível 4 do modelo ISO, estão os protocolos responsáveis por estabelecer uma comunicação entre dois programas de aplicação, garantindo que as mensagens sejam encaminhadas ao usuário correto no computador destino. Estes protocolos relacionam cada mensagem ao seu programa de aplicação destino através de "ports", que são os identificadores dos programas. Estes "ports" são estabelecidos para si pelos próprios programas de aplicação. Assim, em cada mensagem enviada pelo nível de transporte deve estar indicado o "port" do programa destino. Como a relação entre dois programas de aplicação é, em geral, cliente-servidor (ou mestre-escravo), o "port" do servidor deve ser conhecido a priori, para que a comunicação possa ser iniciada. Já o "port" do cliente deve ser enviado junto com a primeira mensagem para que a resposta enviada pelo servidor seja entregue ao cliente. Desta forma, as mensagens recebidas são colocadas à disposição do programa de aplicação associado ao "port" especificado na mesma, enquanto as mensagens a serem enviadas são encaminhadas ao nível de interconexão, como mostra a figura II.4. Dois exemplos de protocolos utilizados neste nível são o TCP (Transmission Control Protocol) [17] e o UDP (User Datagram Protocol) [18].

O protocolo TCP utiliza um modo de comunicação conhecido por "connection mode", no qual dois programas de aplicação de dois computadores ficam interligados, formando o que é conhecido por circuito virtual. Neste modo de comunicação as mensagens enviadas têm um destino pré-definido pela conexão estabelecida. Além disso o TCP é responsável por fragmentar mensagens grandes antes destas serem enviadas, ordenar as diversas partes de uma mensagem recebida, e detectar e reenviar mensagens perdidas.

Já o UDP utiliza outro modo de comunicação, o "connectionless mode", no qual não há uma conexão, ou seja, as mensagens enviadas podem ter destinos diferentes, devendo, portanto, ser explicitamente endereçadas. As mensagens enviadas deste modo são chamadas de datagramas. Ao contrário do TCP, o UDP não garante que datagramas enviados sejam recebidos corretamente, deixando este controle a cargo do programa de aplicação.

II.3. Nível de Interconexão

Neste nível, correspondente ao nível 3 do modelo ISO, está o protocolo IP (Internet Protocol) [19] cuja principal característica é a de tornar transparente para o usuário a topologia da rede. O protocolo IP é responsável pelo percurso das mensagens ao longo da "internet", sendo este percurso baseado no endereço do destino de cada mensagem. Neste nível é, assim, decidido se uma mensagem

será encaminhada diretamente ao seu destino (caso em que a origem e o destino estão na mesma rede) ou se será enviada a um "gateway". Nos computadores intermediários ("gateways"), as mensagens recebidas são retransmitidas para o destino, como mostra a figura II.4, ou para outro "gateway". No computador destino, a mensagem deve ser encaminhada ao protocolo correto no nível superior (o TCP ou o UDP, por exemplo).

É importante diferenciar os dois identificadores de um programa de usuário; um deles é o seu "port" (nível de transporte) e o outro é o endereço na "internet" do computador no qual ele está executando (nível de interconexão). Da mesma forma como ocorre com o "port" no nível de transporte, o endereço na "internet" do computador, no qual está executando o servidor, deve ser conhecido pelo cliente a priori para que a comunicação possa ser iniciada.

II.4. Nível de Interface com a Rede Local

Este nível, correspondente ao nível 2 do modelo ISO, é responsável pela transmissão efetiva das mensagens, possuindo os protocolos necessários à manipulação do meio físico. Estes protocolos são específicos para o meio físico utilizado na comunicação. Um exemplo destes protocolos é o utilizado pela Ethernet [12].

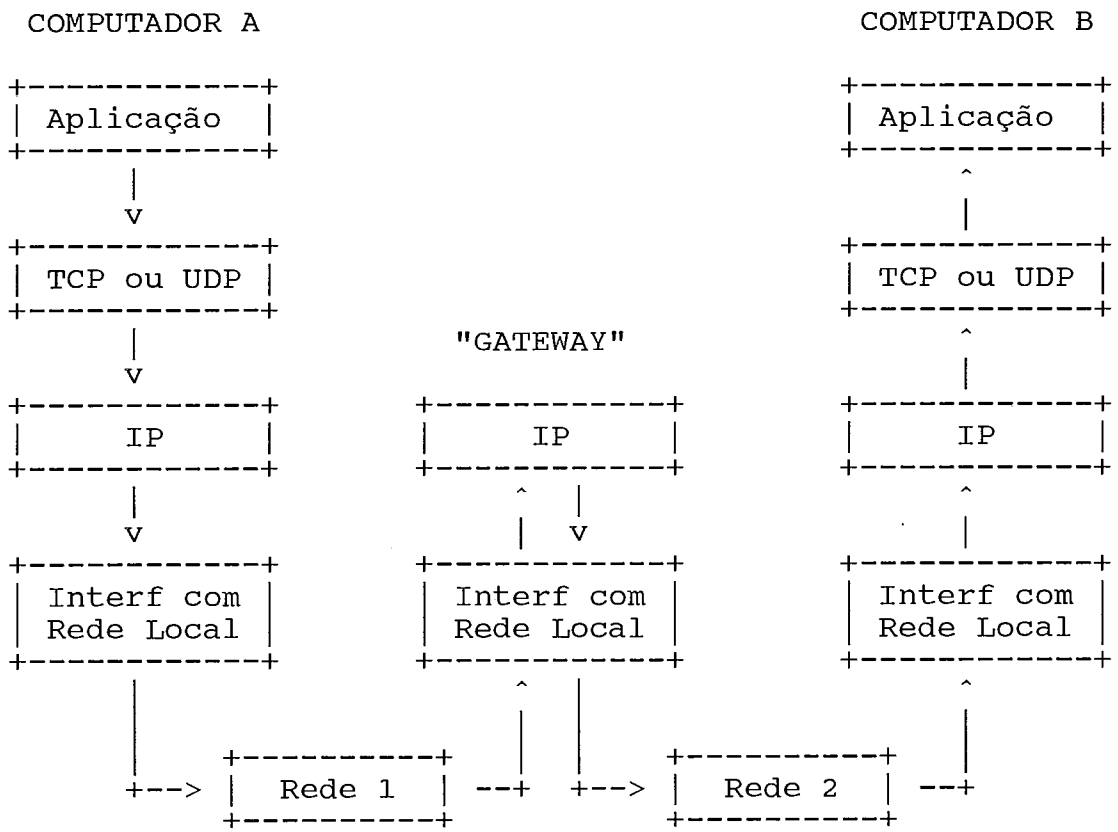

Figura II.4

Percurso de uma mensagem entre uma aplicação do computador A, localizado na Rede 1, e uma do computador B, localizado na Rede 2.

CAPÍTULO III

A INTERFACE DO NÍVEL DE TRANSPORTE

A implementação dos protocolos TCP/IP em um sistema operacional implica na definição de uma interface entre os protocolos e os programas de aplicação que os utilizam. Esta interface se localiza no nível de transporte e é responsável pela comunicação entre este nível e o de aplicação.

Para que os programas de aplicação utilizem os protocolos de comunicação é necessário que haja uma interface que permita estabelecer "ports", criar uma conexão (circuito virtual) entre dois programas de aplicação de dois computadores, transmitir dados (via conexão ou datagrama) entre programas de aplicação e desfazer uma conexão existente.

As interfaces existentes se diferenciam por executar as tarefas descritas acima de modos particulares. As definições de interfaces mais utilizadas nos sistemas "UNIX-like" são a interface do 4.2 BSD UNIX (da Universidade de Berkeley, Califórnia) e a interface do UNIX System V.3 (da AT&T), denominada TLI (Transport-Level Interface).

A interface do 4.2 BSD UNIX [6] é baseada nos chamados "sockets", que permitem ao usuário construir aplicações poderosas utilizando a rede. Um "socket" é um canal de comunicação entre o programa de aplicação e o protocolo utilizado, ou seja, é exatamente o meio pelo qual o programa de aplicação interage com o protocolo.

A TLI [20] é baseada nos "Streams" [5]. Esta interface é tão poderosa quanto os "sockets" e, ainda se mostra mais flexível com relação aos protocolos utilizados, ou seja, alterações nos protocolos não implicam em alterações na interface, como é mostrado por MANNAS [21], que faz uma comparação entre a TLI e os "sockets". Na TLI são utilizados dispositivos especiais, como os dispositivos de E/S, e a comunicação entre o programa de aplicação e o protocolo utilizado é feita através de referências a estes dispositivos.

Mais recentemente, o consórcio X/OPEN (*) padronizou a XTI (X/OPEN Transport Interface) que é derivada da TLI. A XTI [22] define a existência de "endpoints", que são canais de comunicação através dos quais os programas de aplicação utilizam a interface. No entanto, o método de criação e a própria forma dos "endpoints" é dependente da implementação.

As interfaces definem rotinas através das quais os

(*) X/OPEN é Marca Registrada da X/OPEN Company Limited.

usuários obtêm o acesso aos seus serviços. As rotinas disponíveis nas três interfaces apresentadas, em geral, possuem funções similares, apenas utilizando parâmetros diferentes. Sendo assim, ao longo da explicação sobre as interfaces, as referências serão feitas às rotinas relativas aos "sockets", enquanto as rotinas da TLI - as rotinas similares da TLI e da XTI possuem nomes iguais, os quais são sempre precedidos por "t_" - aparecerão logo a seguir, entre parêntesis.

As interfaces são organizadas em 4 partes que representam as diversas fases da comunicação. Estas fases são as seguintes:

- Início e Fim da Utilização de um "endpoint"

  Esta é uma fase local, ou seja, nenhuma informação é enviada pela rede. As funções são responsáveis por iniciar e terminar a utilização da interface propriamente dita.

  É nesta fase que se encontra a diferença básica entre as três interfaces citadas anteriormente, que é a maneira de criação do "endpoint". No 4.2 BSD UNIX, um "socket" ("endpoint") é criado com as características desejadas pelo usuário (rotina socket). Na TLI, um dispositivo especial é aberto (rotina t_open) e as características dele são utilizadas na criação do "endpoint". Na XTI, um "endpoint" é criado (rotina t_open), porém, o método a ser utilizado na sua criação não é definido. Nos três casos, um

identificador retornado será utilizado em todas as outras rotinas para especificar o "endpoint".

Ainda nesta fase estão as rotinas bind (t_bind) que associa um identificador ("port") ao "endpoint", unbind (t_unbind) que libera um "port" associado anteriormente e close (t_close) que libera um "endpoint", além de terminar a conexão (se for "connection mode").

- Conexão

Nesta fase estão as rotinas responsáveis por estabelecer uma conexão (ou circuito virtual) entre duas aplicações. Estas rotinas não são necessárias às aplicações que utilizam o "connectionless mode".

Na criação da conexão um usuário será ativo (cliente) e iniciará a comunicação, enquanto o outro será passivo (servidor) e estará a espera de um pedido de conexão. O "port" do servidor deve ser conhecido pelo cliente, enquanto o "port" do cliente será enviado ao servidor junto com o pedido de conexão. O servidor utiliza a rotina listen (t_listen) para esperar uma conexão, enquanto o cliente pede uma conexão através da rotina connect (t_connect), que envia o pedido ao servidor. Quando o servidor recebe o pedido de conexão, ele o aceita pela rotina accept (t_accept), que envia a resposta (aceitação) ao cliente, estabelecendo a conexão.

O servidor pode estabelecer a conexão através do

mesmo "endpoint" que foi utilizado para receber o pedido, ou criar um novo "endpoint" para a comunicação, liberando o inicial para continuar recebendo pedidos. No caso em que um novo "endpoint" é criado, seu "port" é enviado ao cliente junto com a aceitação do pedido de conexão.

- Transferência de Dados

Nesta fase é feita a troca de mensagens entre dois usuários. No "connection mode", as mensagens são transmitidas pela conexão estabelecida anteriormente com total garantia de chegada na ordem certa e sem erros. Para isso são utilizadas as funções send (t_snd) e recv (t_rcv) que, além de enviar e receber mensagens normais, manipulam mensagens urgentes que podem interromper o fluxo normal de dados entre dois "endpoints". As "system calls" write e read normais do sistema operacional também podem ser utilizadas para enviar e receber mensagens pelos "sockets" ou pela TLI. A XTI, no entanto, não define a utilização destas rotinas.

No "connectionless mode", as mensagens são transmitidas entre dois "endpoints" quaisquer e sua chegada sem erros ao destino não é garantida. Neste modo são utilizadas as rotinas recvfrom (t_rcvudata) e sendto (t_sndudata).

- Término da Conexão

Esta fase só ocorre no "connection mode" e é nela que

ocorre o fim da conexão entre dois "endpoints". Esta desconexão pode ser feita de uma forma ordenada ("orderly") ou abrupta ("abrupt"). Na forma ordenada os dados em trânsito no momento da desconexão serão entregues. Nesta forma são utilizadas as rotinas shutdown para os "sockets" e t_sndrel na TLI e na XTI.

Na forma abrupta os dados em trânsito no momento da desconexão não terão sua chegada ao destino garantida. Nesta forma são utilizadas as rotinas close para os "sockets" e t_snddis (ou t_close) na TLI e na XTI.

As rotinas close e t_close, além de terminar a conexão, liberam o "endpoint".

CAPÍTULO IV

A IMPLEMENTAÇÃO DO SISTEMA DE COMUNICAÇÕES

Como foi mostrado no capítulo II, a família de protocolos TCP/IP pode ser dividida em quatro níveis conceituais: aplicação, transporte, interconexão e interface com a rede local. Na implementação dos protocolos TCP/IP no PLURIX, consideramos como sistema de comunicações apenas os níveis de transporte, de interconexão e de interface com a de rede local. Estes níveis constituem um sistema básico de comunicações, e juntos são capazes de tornar disponíveis, aos usuários do sistema operacional, os serviços de comunicação oferecidos pela rede. O acesso a estes serviços de comunicação é feito através da criação de programas, ou da utilização de produtos disponíveis, que executem os protocolos de aplicação. Desta forma, os protocolos do nível de aplicação são, na realidade, executados por programas de aplicação que utilizam os serviços da rede através do sistema de comunicações disponível no sistema operacional.

O sistema de comunicações do PLURIX, formado, basicamente, pelas rotinas que executam os protocolos de mais baixo nível (interface com a rede local, interconexão e transporte), foi instalado dentro do núcleo do sistema operacional. Esta decisão se deveu a dois fatores

principais: performance e existência, dentro do núcleo, de mecanismos úteis ao sistema de comunicações, como a manipulação de "timeouts", por exemplo. Outra opção seria a sua colocação em um ou mais processos de usuário [23], mas isto, além de ineficiente, seria dificultado pelo fato de o PLURIX não fornecer aos processos de usuário um mecanismo eficiente para a comunicação entre processos. É importante ressaltar que em um sistema de comunicações uma grande quantidade de dados é manipulada. Estes dados, para serem transmitidos ou recebidos, passam necessariamente pelo núcleo do sistema operacional. Caso o sistema de comunicações estivesse em um processo de usuário, estes dados deveriam, ao ser recebidos, por exemplo, passar do núcleo do sistema para o processo do sistema de comunicações, para, só então, passar ao processo de aplicação. No caso em que o sistema de comunicações se encontre dentro do núcleo, é ele mesmo que recebe os dados, passando-os para o processo de aplicação.

Um aspecto importante da execução do sistema de comunicações implementado é que, embora instalado dentro do núcleo do sistema, ele é executado através de processos independentes, cujo funcionamento será explicado na seção IV.2. Como menciona CLARK [23], este esquema, embora atraente por não causar transformações no núcleo do sistema, não é muito utilizado na construção de sistemas de comunicações em razão da não disponibilidade nos sistemas operacionais de processos que executem dentro do seu núcleo. Como o PLURIX foi construído para ser

executado em um ambiente multiprocessado, não há empecilhos para que diversos processos possam executar concorrentemente dentro do seu núcleo. Desta forma, este esquema foi utilizado com êxito, permitindo assim, que o sistema de comunicações fosse construído como uma parte incremental dentro do núcleo do PLURIX.

Como foi visto, a execução dos protocolos de aplicação é realizada por processos de usuário, enquanto o sistema de comunicações executa em modo supervisor. Por isso é necessário que haja uma interface entre os dois, para que os programas de aplicação possam utilizar os serviços oferecidos pelo sistema de comunicações. Esta interface fica entre os níveis de transporte e o de aplicação, e, como foi mostrado no capítulo III, é chamada de interface de transporte. É nesta interface que se encontra o ponto de entrada do sistema de comunicações (dentro do núcleo do sistema operacional).

IV.1. A Interface do Nível de Transporte

Na implementação do TCP/IP no PLURIX, foi escolhida, como padrão para a interface do nível de transporte, a XTI (X/OPEN Transport Interface) [22] - padrão de interface definido pelo X/OPEN. Esta escolha foi baseada no fato de que, por ser totalmente independente dos detalhes de implementação das diversas versões de sistemas "UNIX-like" existentes, ela poderia ser implementada no PLURIX, sem

que fossem necessárias alterações na filosofia do sistema.

Como foi mostrado no capítulo III, a XTI define a existência de "endpoints", através dos quais os programas de aplicação utilizam a interface. Estes "endpoints" devem ser manipulados pelo usuário através das rotinas definidas pela interface. No entanto, o método de criação e a própria forma dos "endpoints" é dependente da implementação.

A XTI permite que dois modos de comunicação sejam usados, o "connection mode" e o "connectionless mode", garantindo, assim, a possibilidade de transferência de mensagens em circuito virtual e a transmissão de datagramas. Desta maneira, a XTI define rotinas que possibilitam a manipulação local do "endpoint", e que realizam a conexão, a desconexão, a transferência de mensagens em um circuito virtual e a transmissão de datagramas.

A seguir serão mostradas as rotinas pertencentes à definição da XTI e suas respectivas funções.

- Manipulação Local de um "endpoint"

t_open	Cria um "endpoint".
t_bind	Associa um identificador ("port") ao "endpoint".
t_unbind	Libera o "port" associado ao "endpoint".
t_close	Termina o uso de um "endpoint".

t_error	Gera uma mensagem de erro. (opcional)
t_alloc	Aloca uma área para uma estrutura especificada. (opcional)
t_free	Libera uma área alocada pela t_alloc. (opcional)
t_look	Retorna o evento ocorrido.
t_sync	Retorna o estado do "endpoint".
t_getinfo	Obtém informações sobre um protocolo específico. (opcional)
t_getstate	Obtém o estado corrente do "endpoint". (opcional)
t_optmgmt	Manipula opções. (opcional)

- Conexão

t_connect	Estabelece uma conexão com outro usuário.
t_rcvconnect	Recebe a aceitação de um pedido de conexão.
t_listen	Espera por um pedido de conexão.
t_accept	Aceita um pedido de conexão.

- Transferência de Dados

t_snd	Envia dados, urgentes ou não, através de uma conexão.
t_rcv	Recebe dados, urgentes ou não, através de uma conexão.
t_sndudata	Envia um datagrama.
t_rcvudata	Recebe um datagrama.
t_rcvuderr	Recebe a indicação de erro no envio de um datagrama.

- Término da Conexão

t_snddis	Envia um pedido de desconexão.
t_rcvdis	Recebe a indicação de um pedido de desconexão.
t_sndrel	Envia um pedido de desconexão ordenada. (opcional)
t_rcvrel	Recebe a indicação de pedido de desconexão ordenada. (opcional)

A XTI manipula os "endpoints" utilizando para eles 8 estados. O usuário pode, a cada momento, saber em que estado se encontra um "endpoint", através da rotina t_sync. Isto torna possível que um "endpoint" seja compartilhado por processos distintos. Os estados definidos pela XTI são os seguintes:

T_UNINIT	Estado inicial e final de um "endpoint" (desativado).
T_UNBND	O "endpoint" não está associado a nenhum identificador ("port").
T_IDLE	Não há conexões com este "endpoint".
T_OUTCON	Foi enviado um pedido de conexão.
T_INCON	Há pedidos de conexão pendentes para este "endpoint".
T_DATAXFER	O "endpoint" está transferindo dados.
T_OUTREL	Foi enviado um pedido de desconexão.
T_INREL	Há um pedido de desconexão pendente para este "endpoint".

As tarefas realizadas por uma interface no nível de

transporte são essencialmente assíncronas, ou seja, vários eventos podem ocorrer independentemente das ações do usuário. Por exemplo, uma mensagem de desconexão de um circuito virtual pode chegar no momento em que o usuário está utilizando este circuito para enviar mensagens. A XTI define dois modos de execução para manipular eventos assíncronos, o modo síncrono e o modo assíncrono. No modo síncrono, as rotinas da interface aguardam eventos específicos antes de retornar o controle ao usuário. Enquanto espera, o usuário não pode realizar outras tarefas. No modo assíncrono, um mecanismo é utilizado para informar o usuário sobre a ocorrência de algum evento, sem que o usuário espere por ele. A manipulação assíncrona de eventos é uma característica importante em uma interface do nível de transporte, pois possibilita que os usuários realizem outras tarefas, enquanto aguardam a ocorrência de um determinado evento.

Os eventos assíncronos definidos na XTI são:

T_LISTEN	Foi recebido um pedido de conexão de um usuário remoto.
T_CONNECT	Foi recebida a resposta a um pedido de conexão enviado anteriormente.
T_DATA	Foi recebida uma mensagem normal.
T_EXDATA	Foi recebida uma mensagem urgente.
T_DISCONNECT	Foi recebido um pedido de desconexão.
T_ORDREL	Foi recebido um pedido de desconexão ordenada.
T_UDERR	Foi detectado um erro em um datagrama

enviado anteriormente.

T_GODATA — O fluxo de dados está normalizado. Uma nova mensagem pode ser enviada.

T_GOEXDATA — O fluxo de dados está normalizado. Uma nova mensagem urgente pode ser enviada.

Alguns destes eventos são específicos para um determinado modo de comunicação. Sendo assim, programas que utilizam o "connection mode" não necessitam tratar o evento T_UDERR, e programas que utilizam o "connectionless mode" não necessitam tratar os eventos T_LISTEN, T_CONNECT, T_EXDATA, T_DISCONNECT, T_ORDREL e T_GOEXDATA.

Um usuário verifica a ocorrência de eventos assíncronos através da chamada à rotina t_look, que retorna o evento ocorrido. Além disso, o erro TLOOK é retornado pelas diversas rotinas da interface, caso um evento inesperado tenha ocorrido. Desta maneira, mesmo as aplicações que utilizam o modo síncrono, podem reconhecer os eventos assíncronos e tratá-los caso seja necessário.

No PLURIX, como foi visto, a melhor solução encontrada para a implementação dos protocolos TCP/IP, foi a colocação das rotinas que executam os protocolos dos níveis de transporte, interconexão e interface com a rede local, dentro do núcleo do sistema operacional. Por esta razão, a XTI, no PLURIX, deve exercer a função de ligação entre o modo usuário (nível de aplicação) e o modo supervisor (níveis mais baixos). A implementação da

interface XTI foi, assim, realizada utilizando-se a estrutura já existente no sistema para o tratamento dos dispositivos especiais de caracteres, os "drivers" de E/S [5].

Um "driver" é formado pelas rotinas open, close, read, write e ioctl específicas para um determinado dispositivo. Isto significa que estas cinco "system calls" (chamadas ao sistema) identificam o dispositivo para o qual foram chamadas e desviam para a rotina específica do "driver" correspondente (dentro do núcleo do sistema).

Sendo assim, foi criado um "driver" para cada família de protocolos (Internet, por exemplo), e dentro de cada um destes "drivers" foi colocado o tratamento dos diversos protocolos que cada família possui (TCP e UDP - protocolos Internet, por exemplo). Além disso, para cada protocolo (de cada família) foi criado, no sistema de arquivos, um dispositivo de caracteres sem nenhum atributo físico. Estes dispositivos possuem dois identificadores que os associam aos protocolos e, por conseguinte, aos "drivers". Eles são o "major" que indica a família de protocolos, e o "minor" que indica o protocolo propriamente dito.

Por ser a ligação entre os níveis de aplicação e transporte, e por estes se localizarem respectivamente fora e dentro do núcleo do sistema operacional, a interface de transporte deve permanecer tanto fora quanto dentro do núcleo. Sendo assim, as rotinas da interface XTI

estão disponíveis aos usuários em uma biblioteca específica (modo usuário). Estas rotinas chamam as rotinas do sistema, que desviam para as rotinas do "driver" correspondente ao dispositivo especificado (modo supervisor), ou mais especificamente, para uma rotina interna do sistema, correspondente à rotina da interface. A especificação do dispositivo é feita através da rotina t_open que recebe, como parâmetro, o seu nome. Este nome é passado à "system call" open, que executa o desvio à rotina open do "driver" correspondente ao dispositivo. A open específica cria o "endpoint", que é uma estrutura contendo as informações a serem usadas na comunicação, de acordo com o protocolo especificado pelo dispositivo. No final, a t_open retorna ao usuário um descritor que deve ser usado como argumento nas chamadas a todas as rotinas da interface, a fim de especificar o "driver" e o "endpoint". Desta maneira, um usuário pode manipular vários "endpoints", mesmo que eles sejam correspondentes ao mesmo protocolo.

É importante ressaltar que a utilização dos "drivers" na implementação da interface fez com que não fosse necessária a criação de novas "system calls", ao contrário do que foi feito no 4.2 BSD UNIX [24]. Todas as rotinas da interface executam uma "system call" já existente no sistema (open, close, read, write ou ioctl), que desvia para uma rotina de um dos "drivers". A ioctl é utilizada pela maioria das rotinas, que se identificam através de um dos parâmetros. Este parâmetro é posteriormente utilizado

no desvio para a rotina correta dentro do "driver".

Uma pequena alteração no sistema de "drivers" foi, contudo, necessária à sua utilização no sistema de comunicações. No PLURIX, a "system call" close só desvia para a close do "driver" correspondente ao dispositivo a ser fechado no caso de não haver mais nenhum processo (ou até outra instância do mesmo processo) utilizando o dispositivo em questão. No caso especial dos "endpoints", um mesmo dispositivo é utilizado para todos os "endpoints" que utilizem o mesmo protocolo. Como a close dos "drivers" relativos aos protocolos é responsável por desativar um "endpoint", liberando-o para outros usuários, ela deve ser chamada sempre que uma close ("system call") for utilizada para um dispositivo deste tipo. Por isso, foi criado, na tabela de "drivers" do sistema, um campo com um "flag" que indica se, para determinado dispositivo, a close do "driver" deve ou não ser chamada sempre.

IV.2. A Organização dos Protocolos

Na implementação do sistema de comunicações do PLURIX foi feita a opção pela divisão dos diversos protocolos, dos diversos níveis, em rotinas, criando assim uma modularização funcional. Esta forma de implementação é bastante interessante por garantir a versatilidade necessária a uma possível substituição ou inclusão de protocolos no sistema. No entanto, como o total isolamento

entre os diversos níveis pode ser prejudicial ao desempenho do sistema, como mostra CLARK [23], houve a preocupação de fornecer, às rotinas de cada nível, informações acerca do funcionamento das rotinas dos demais níveis. Um exemplo destas informações é o tamanho máximo do cabeçalho do protocolo IP, que deve ser conhecido pelas rotinas do nível de transporte, para que estas lhe reservem espaço no "buffer" de cada mensagem. A forma como estas rotinas interagem entre si, bem como a eficiência resultante, são determinadas pela organização do sistema de comunicações.

Os "endpoints", já citados como membros de interligação entre os programas de aplicação e a interface de transporte, são também utilizados na interação entre esta e os protocolos do nível de transporte (TCP e UDP, por exemplo). Assim, como é notório o fato de que toda a organização do sistema se baseia nos "endpoints", é importante ressaltar algumas de suas características.

Para um melhor entendimento da descrição dos "endpoints" e também das seções seguintes, é importante estar atento à diferença entre dois elementos básicos do sistema de comunicações. Estes elementos são as mensagens e os dados. Como mostra a figura IV.1, uma mensagem é exatamente um pacote formado pelas informações relativas aos protocolos que a formaram e pelos dados que o usuário deseja enviar ou receber. Os dados são informações sem significado para o sistema de comunicações, cuja função é

simplesmente fornecer um meio para a sua transmissão. Assim, os dados fornecidos pelo usuário são colocados ordenadamente em mensagens, que são enviadas ao destino estabelecido pelo mesmo. Da mesma forma, as mensagens recebidas pelo sistema de comunicações contêm dados, que são ordenadamente colocados à disposição do usuário para o qual foram destinados.

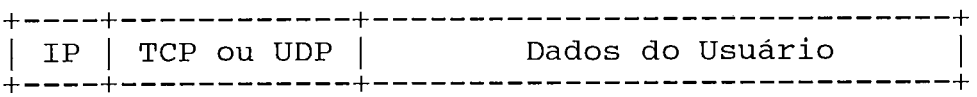

Figura IV.1

Formato das Mensagens

Os "endpoints" se localizam em listas (vetores) de acordo com o protocolo utilizado. Assim, no momento, o PLURIX contém uma lista para o protocolo TCP e uma outra para o protocolo UDP. Em breve, será acrescentada ainda uma lista de "raw-endpoints" que, como os "raw-sockets" [6], permitirão a utilização direta do protocolo IP pelos programas de aplicação.

O acesso aos "endpoints" pela interface de transporte é feito diretamente. Seu índice na lista consta em uma estrutura relacionada ao seu descritor, e a lista é indicada pelo "minor" do dispositivo utilizado. Já o acesso pelos protocolos de transporte, na recepção de uma mensagem, é feito, como no sistema operacional XINU [11], através de uma busca seqüencial realizada na lista do

respectivo protocolo. A chave utilizada nesta busca é o "port" especificado em cada mensagem. Está prevista para futuras extensões do sistema de comunicações a agilização desta busca. Uma opção é a incorporação de um "cache", como é sugerido por WATSON & MAMRAK [7]. Neste caso particular, o êxito na utilização do "cache" se deve a existência de um "working set" bem definido das conexões ativas. Outra opção ainda mais eficiente é a utilização de algoritmos de "hash".

Os "endpoints" utilizados pelo UDP, se diferenciam dos utilizados pelo TCP por serem mais simples. Cada um deles contém, além de informações necessárias à comunicação, uma fila de mensagens recebidas (fila de entrada). Já os "endpoints" utilizados pelo TCP, contêm, além das informações necessárias à comunicação, informações necessárias à conexão, uma fila de mensagens recebidas (fila de entrada), uma fila de mensagens a serem enviadas (fila de saída) e uma fila de mensagens a serem retransmitidas (fila de retransmissão). As mensagens recebidas, nos dois casos, esperam por uma requisição do usuário. As mensagens a serem enviadas, esperam por espaço na janela de recepção (número de bytes de dado que um "endpoint" pode receber antes que um "ack" seja enviado) [17] do "endpoint" destino, ou seja, esperam que o usuário destino possa receber os dados nelas contidos. As mensagens a serem retransmitidas esperam por um "ack" (depois do qual são retiradas da fila) ou por um "timeout" (depois do qual são retransmitidas e recolocadas na fila).

Para um melhor entendimento da organização do sistema de comunicações, sua explicação será feita em duas partes. Em um primeiro passo, será mostrado o envio de uma mensagem desde sua fonte, o programa de aplicação, até sua colocação na linha de comunicação. Em um segundo passo, será descrito o caminho inverso da mensagem, desde sua chegada pela linha até seu recebimento pelo programa de aplicação. Estes trajetos são ilustrados, respectivamente, pelas figuras IV.2 e IV.3.

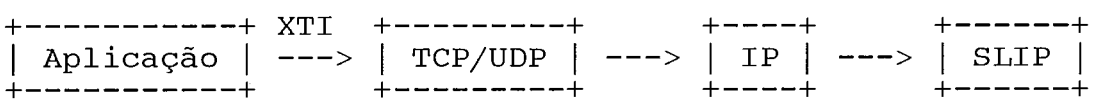

Figura IV.2

Trajeto percorrido por uma mensagem a ser enviada

O envio de uma mensagem é totalmente realizado pelo processo do usuário. Este mesmo esquema é utilizado no sistema operacional XINU [11]. Como mostra a figura IV.2, o acesso aos serviços de comunicação é feito através de uma das rotinas da interface (rotina t_snd para enviar dados, por exemplo) que entra no modo supervisor através do "driver" especificado pelo descritor do "endpoint". Assim, já em modo supervisor, o próprio processo do usuário verifica as condições do "endpoint", passando então o endereço dos dados para o nível de transporte. É neste nível que os dados do usuário são copiados para "buffers" do sistema, e são montadas as mensagens (TCP ou UDP), já com o cabeçalho específico do protocolo. No caso

do protocolo UDP, a mensagem é passada diretamente ao próximo nível, o de interconexão. Já no caso do protocolo TCP, a mensagem só será enviada caso haja espaço na janela do destino. Isto significa que se o destino não puder receber a mensagem naquele momento e se não couber mais mensagens na fila de saída do "endpoint", o processo ficará esperando (modo síncrono), ou não (modo assíncrono), por um evento (espaço na janela ou na fila). Além disso, cada mensagem TCP enviada ao nível de interconexão é colocada também em uma fila de retransmissão. E, no caso de não chegar um "ack" depois de um certo intervalo de tempo ("timeout"), a mensagem é automaticamente reenviada.

No nível de interconexão, onde se encontra o protocolo IP, a mensagem recebida do nível superior é completada com o cabeçalho deste protocolo. Neste nível, é feita a fragmentação da mensagem caso seja necessário, e após ser decidido o seu destino, ela é encaminhada ao nível seguinte, o nível de interface com a rede local. O protocolo usado no nível de interface com a rede local depende do meio físico utilizado na comunicação. Como, por enquanto, está sendo utilizada uma linha serial, o nível de rede está usando o protocolo SLIP (Serial Line Protocol) [13]. Neste nível a mensagem é enviada efetivamente.

O fato de estar sendo utilizada uma linha serial possibilitou a implementação de um IP com funções

reduzidas. No algoritmo de roteamento, por exemplo, só existem duas possibilidades, o envio da mensagem pela linha de comunicação ou pelo modo interno. Este modo interno, definido inicialmente para testar os protocolos, consiste na utilização como meio de comunicação de uma fila de mensagens. Assim, mensagens cujo processo destino se encontra no mesmo computador que o processo remetente, não saem do sistema. Este tipo de mensagem tem, assim, sua transmissão otimizada, além de não ocupar a rede inutilmente. O modo interno de comunicação é eficiente por que as mensagens não são transmitidas; cada uma é copiada para um novo "buffer", que é simplesmente colocado na fila. Assim, o envio de uma mensagem pelo modo interno é equivalente a sua entrada nesta fila.

Já o recebimento e tratamento de mensagens é totalmente realizado por um processo específico chamado "daemon". Este processo é um programa de usuário, que executa em modo supervisor e é responsável por toda a manutenção do sistema de comunicações. Sua execução deve ser iniciada antes que qualquer outro programa tente utilizar os serviços de comunicação, já que é ele que inicia o sistema, viabilizando a comunicação. Sua entrada no modo supervisor é feita através de um dispositivo próprio e o seu término, por interrupção ou por erro, determina o fim da utilização do sistema de comunicações.

O "daemon", além de ser responsável pela retransmissão de mensagens que não receberam "ack",

monitora a linha de comunicação e a fila do modo interno, esperando mensagens. Uma rotina (que executa o protocolo SLIP) recebe as mensagens que chegam na linha de comunicação, coloca-as em "buffers" e encaminha cada uma ao protocolo de interconexão para o qual foi destinada (o IP, por exemplo). No nível de interconexão, o endereço do destino da mensagem é verificado e ela é encaminhada ao protocolo do nível seguinte especificado na mesma (UDP ou TCP, por exemplo). É também no nível de interconexão, que é feito o reagrupamento das mensagens que chegaram fragmentadas. Uma mensagem só passa para o nível superior, depois que todos os seus fragmentos tenham chegado.

No nível de transporte, o tratamento da mensagem é diferente para cada protocolo. Para o UDP, os dados recebidos na mensagem são simplesmente colocados à disposição do usuário na fila de entrada do "endpoint" especificado na mensagem pelo "port". Já para o TCP, o tratamento das mensagens que chegam é bem mais complexo. Como as mensagens TCP podem chegar fora de ordem e o usuário deve recebê-las na ordem correta, o sistema deve se preocupar em ordená-las antes de colocá-las à disposição do usuário. Assim, cada mensagem é inserida na fila de entrada de acordo com seu número de seqüência. Uma mensagem TCP só está disponível ao usuário depois que todas as mensagens com número de seqüência menor que o dela estiverem disponíveis. Além disso, o TCP faz um controle de mensagens recebidas corretamente pelo envio de "acks" [17]. O recebimento do "ack" de uma mensagem faz

com que todas as mensagens com número de seqüência menor ou igual ao dela sejam retiradas da fila de retransmissão. Junto com o "ack" de uma mensagem, podem ser enviados dados que anteriormente não couberam na janela e que por isso estavam aguardando na fila de saída. Este controle também é realizado pelo "daemon".

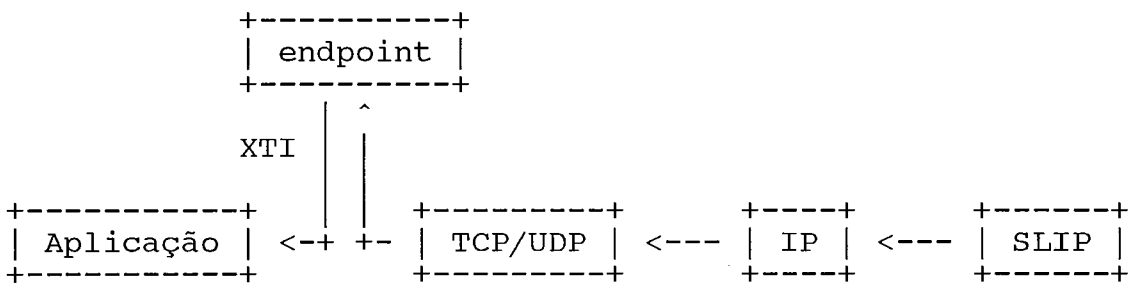

Figura IV.3

Trajeto percorrido por uma mensagem recebida

Como mostra a figura IV.3, o "daemon" é responsável por colocar cada mensagem recebida na fila de entrada do "endpoint" especificado no "port" da mesma. Estas mensagens serão retiradas da fila pela interface, mediante a requisição do usuário. Desta forma, estando disponível em um "endpoint", uma mensagem pode ser lida, através da interface, pelo usuário associado ao mesmo. Há, assim, dois processos que se comunicam através dos "endpoints", o processo do usuário e o processo que trata a chegada de mensagens.

É importante ressaltar que este esquema permitiu que os dados fossem copiados apenas uma vez no envio e uma vez

na recepção de mensagens. Em ambos os casos, a cópia é feita pelo processo do usuário. No envio, a cópia é feita da área do usuário para a do sistema, no nível de transporte. Na recepção, a cópia é feita da área do sistema para a do usuário, pela interface de transporte. Isto é bastante significativo para a performance do sistema já que, em geral, a quantidade de dados manipulada é muito grande.

O tratamento dos protocolos TCP/IP é, em resumo, feito por processos distintos. Na entrada, o tratamento é feito por um processo específico do sistema, responsável pela recepção das mensagens. Na saída, o tratamento é feito pelo próprio processo do usuário, embora seja executado dentro do sistema. Esta divisão das funções pelos processos, visto que a cada processo é atribuída a tarefa de enviar suas próprias mensagens, contribui para um melhor desempenho do sistema. Se a tarefa de enviar mensagens fosse realizada pelo "daemon", este estaria realizando-a não para um usuário, mas para todos, acarretando um possível atraso na recepção de mensagens. Outro artifício para agilizar a execução do "daemon" é utilizado na manipulação das filas dos "endpoints". Estas filas são organizadas de tal forma que, seja qual for a ação (inserção ou remoção) realizada pelo "daemon", ela é sempre realizada no primeiro "buffer" da fila. Assim a tarefa de percorrer as filas é sempre atribuída ao processo do usuário. Esta manipulação das filas será explicada com detalhes em uma seção posterior.

IV.3. Mecanismos Básicos Utilizados

Nas seções iniciais deste capítulo foram apresentados os módulos principais do sistema de comunicações implementado no PLURIX. Nesta seção serão mostrados alguns mecanismos básicos que propiciaram esta implementação.

IV.3.1. Os "buffers" de Dados

Os sistemas de comunicações têm basicamente a função de prover o sistema operacional de um mecanismo de transferência de mensagens. Sendo assim, a principal estrutura de dados em que se baseia um sistema de comunicações é aquela na qual as mensagens são armazenadas antes de serem enviadas ou depois de serem recebidas. Esta estrutura é o "buffer" de dados, que por ser de importância vital ao sistema, deve ser projetado de forma que sua manipulação seja extremamente eficiente.

Os "buffers" de dados do sistema de comunicações do PLURIX fazem parte de um "pool de buffers" independente dos "buffers" do sistema. Isto permitiu que eles fossem projetados de forma a garantir eficiência na sua manipulação. O "pool de buffers" do sistema de comunicações consiste de uma lista encadeada de "buffers", os quais, além de possuir a área destinada às mensagens

(cada "buffer" armazena exatamente uma mensagem), possui dois ponteiros para outros "buffers". Um destes ponteiros é o responsável pela formação do próprio "pool" e das listas de mensagens contidas nos "endpoint". O outro é um ponteiro auxiliar utilizado para tornar viável certos movimentos nestas listas.

A manipulação do "pool de buffers" é feita por duas rotinas específicas: a getbuf e a freebuf. Quando um processo necessita um "buffer", este é retirado da frente da lista, sendo recolocado, também na frente da lista, após sua utilização.

Nas filas dos "endpoints", os "buffers" ficam organizados em listas encadeadas pelo mesmo ponteiro utilizado no "pool". Nos "endpoints" UDP, a medida que as mensagens vão chegando, elas vão sendo inseridas pelo "daemon" na frente da fila de entrada, sendo retiradas posteriormente pelo processo de usuário, que se encarrega de percorrer a lista para encontrar a mensagem mais antiga.

Já nos "endpoints" TCP, cada uma das filas (entrada, saída e retransmissão) é manipulada de uma forma específica. A fila de entrada é organizada pela ordenação decrescente do número de seqüência. Como na maior parte das vezes as mensagens chegam em ordem, o "daemon" vai, em geral, inserir as mensagens na frente desta fila. Além disso, o ponteiro auxiliar é utilizado para transformar

esta fila em uma lista duplamente encadeada, com o intuito de facilitar o percurso da lista no momento em que o "daemon" a percorre de trás para frente, a partir da última mensagem colocada, para calcular o "ack" a ser enviado. Este procedimento é necessário para que um "ack" enviado em resposta a uma mensagem se refira também a mensagens posteriores (pelo número de seqüência) que tenham chegado anteriormente.

A fila de saída é organizada ao contrário da de entrada, ou seja, os processos inserem as mensagens no fim da lista. Desta forma, no momento de enviar uma mensagem, o "daemon" não necessita percorrer a lista. A fila de retransmissão é organizada da mesma forma que a de saída, facilitando a remoção de mensagens pelo "daemon". Esta remoção é feita de acordo com os "acks" recebidos. Quando uma mensagem deve ser retransmitida por não ter recebido o "ack" no "timeout" determinado, ela é colocada automaticamente na fila de retransmissão do sistema, onde fica aguardando a retransmissão efetiva. Neste caso, a mensagem não passará de uma fila para outra, ela fará parte das duas, sendo que esta outra fila será formada pelo ponteiro auxiliar.

Este esquema de manipulação das filas de mensagens tem por intuito retirar do "daemon" a função de percorrer as listas. Sendo realizada, assim, por cada processo de usuário, esta tarefa passa a não ser mais tão onerosa ao sistema de comunicações.

Outro aspecto importante da utilização dos "buffers" é a manipulação dos dados neles colocados. Como foi mencionado anteriormente, os dados são movimentados apenas uma vez no envio e uma na recepção. Por isto, uma vez copiados para um "buffer", os dados devem permanecer aí até a sua cópia para a área do usuário (recepção), ou a sua colocação na linha de comunicação (envio efetivo). Assim, todas as operações realizadas com os dados, são na realidade realizadas com ponteiros para os "buffers", que contêm, a cada momento, as informações necessárias à sua manipulação. Uma mensagem recebida, por exemplo, tem não só os seus dados, como também os cabeçalhos dos protocolos utilizados, copiados para um "buffer". Assim, quando no nível de transporte esta mensagem for colocada na fila de entrada do "endpoint", a área ocupada pelos cabeçalhos dos protocolos será utilizada para armazenar informações importantes a serem usadas pelo protocolo (no cálculo de "acks" a serem enviados, por exemplo) e pela interface de transporte (na cópia dos dados para área de usuário). Outro exemplo interessante pode ser visto no envio de mensagens TCP. Os protocolos de transporte utilizam um pseudo-cabeçalho [17-18] para o cálculo do "check sum" de suas mensagens. Este pseudo-cabeçalho, que não é enviado, é colocado no início do "buffer" que será utilizado para armazenar o cabeçalho do protocolo de transporte e os dados da mensagem. No momento do envio da mensagem, o início do mesmo "buffer" é preenchido com o cabeçalho do protocolo IP, para depois acomodar as informações que serão necessárias a uma possível retransmissão. Assim,

após uma pequena alteração no seu conteúdo, este mesmo "buffer" passa da fila de saída para a de retransmissão.

Com o intuito de não aumentar as funções do "daemon", as mensagens na fila de retransmissão ficam prontas para ser enviadas, não necessitando uma reconstrução. No entanto, CLARK [23] sugere que para facilitar a retirada das mensagens desta fila, o que ocorre com mais freqüência que a retransmissão, as mensagens não deveriam ficar armazenadas nos respectivos pacotes. Porém, em geral, os "acks" recebidos são relativos a um ou mais pacotes, tornando a liberação dos dados neles contidos imediata, já que as informações acerca destes dados são facilmente obtidas.

IV.3.2. A Retransmissão de Mensagens

A implementação da retransmissão de mensagens TCP [17] utiliza um mecanismo de tratamento de "timeout" existente no núcleo do sistema operacional. Este mecanismo é ativado por uma rotina (toutset) que coloca em uma fila, de acordo com o intervalo de tempo especificado, o endereço de uma rotina, a ser ativada por uma interrupção no fim deste intervalo de tempo, e dois possíveis parâmetros utilizados na chamada a esta rotina. Há, ainda, uma rotina (toutreset) capaz de desativar o mecanismo, caso o evento esperado tenha ocorrido.

No caso do sistema de comunicações, o mecanismo de "timeout" é ativado quando uma mensagem é enviada e colocada na fila de retransmissão do "endpoint". A rotina, a ser ativada ao fim do intervalo de tempo especificado na ativação do "timeout", é responsável por colocar a mensagem cujo endereço, também especificado na ativação do "timeout", ela recebe como parâmetro, na fila de retransmissão do sistema de comunicações para ser enviada posteriormente pelo "daemon". Quando um "ack" confirma a chegada de uma mensagem no destino, esta é retirada da fila de retransmissão do "endpoint", além de ter seu "timeout" interrompido.

No PLURIX, o mecanismo de "timeouts" não prevê a possibilidade de haver vários "timeouts" associados à mesma rotina a ser ativada na interrupção. Sendo assim, a rotina de desativação do mecanismo de "timeout" baseia sua busca na fila apenas no endereço da rotina. O sistema de comunicações, por sua vez, ativa "timeouts" para mensagens diferentes utilizando sempre a mesma rotina. Por este motivo, foi criada uma rotina alternativa para desativar os "timeouts" relativos às mensagens do sistema de comunicações. Esta rotina baseia sua busca na fila não só no endereço da rotina a ser ativada na interrupção, mas também em um dos parâmetros, no caso, o endereço da mensagem.

Esta rotina alternativa foi também utilizada para desativar o mecanismo de "timeout" utilizado para detectar

o silêncio de um "endpoint" remoto durante uma conexão. O mecanismo de "timeout" é ativado para cada "endpoint" local em fase de transmissão de dados. A cada "ack" recebido, o "timeout" é atualizado, ou seja, desativado e ativado novamente. Se um "timeout" terminar, a rotina de interrupção gera um evento T_DISCONNECT para o "endpoint" local. Como esta rotina é a mesma para todos os "endpoints" locais, cada um deles deve ser associado ao seu "timeout" através do seu endereço, que é utilizado pela rotina alternativa para desativar apenas o mecanismo de "timeout" correspondente ao "endpoint" que recebeu o "ack".

IV.3.3. A Manipulação da Janela e do "Acknowledgement"

A principal característica do protocolo TCP é a correção dos dados entregues ao usuário destinatário, que os recebe exatamente na mesma ordem em que foram enviados pelo usuário remetente. Este controle é realizado através do número de seqüência das mensagens e dos seus respectivos "acks". Cada byte dos dados enviados através de uma conexão possui um número de seqüência, assim como os comandos SYN (início de conexão) e FIN (fim da transmissão de dados) [17]. O número de seqüência de uma mensagem é exatamente o número do primeiro byte do seu segmento de dados.

O envio do "acknowledgement" está intimamente

relacionado aos números de seqüência. Um "ack" enviado possui o valor do número de seqüência do próximo byte de dados esperado, significando que todos os dados com número de seqüência menor foram recebidos corretamente. Assim, o "ack" correspondente a uma mensagem tem um valor equivalente ao número de seqüência da mensagem somado ao tamanho do seu segmento de dados.

Outro mecanismo importante para o funcionamento do protocolo TCP é o controle do fluxo de dados estabelecido pela manipulação da janela, que é o número de bytes que um "endpoint" pode enviar a outro sem que um "ack" tenha sido recebido em resposta às mensagens enviadas.

O bom desempenho de um sistema de comunicações, sobretudo no que se refere à utilização da rede, depende, basicamente, dos mecanismos de manipulação da janela e do "acknowledgement". Estes mecanismos são responsáveis por evitar a ocorrência da síndrome da janela pequena (SWS - Silly Window Syndrome) descrita por CLARK [25]. Esta síndrome se caracteriza pelo envio de mensagens pequenas que sobrecarregam a rede, pois para um mesmo número de dados uma quantidade muito maior de informações de controle tem que ser enviada, e também o sistema de comunicações, pois para um mesmo número de dados o "daemon" tem que ser executado um número muito maior de vezes.

O esquema de envio de dados, implementado na execução

do protocolo TCP do sistema de comunicações do PLURIX, foi determinante para a não ocorrência da síndrome da janela pequena. No esquema implementado, o preenchimento de uma mensagem é baseado no tamanho máximo do segmento de dados TCP estabelecido pelo TCP remoto e não no tamanho da janela corrente. Assim, quando um usuário deseja enviar dados, estes são distribuídos em mensagens de tamanho máximo que são enviadas imediatamente, ou são colocadas na fila de saída para serem enviadas quando houver espaço na janela. Caso, no fim da fila, uma mensagem não tenha tamanho máximo, ela poderá ser completada se no momento que o usuário enviar mais dados ela ainda estiver esperando por espaço na janela. Desta forma, mesmo se momentaneamente a janela for diminuída, mensagens pequenas não serão enviadas, evitando a síndrome. Este esquema tem o mesmo efeito que o artifício sugerido por CLARK [25] de suspender o envio de mensagens se a janela estiver pequena. Porém, a janela é considerada pequena somente se o seu tamanho corrente for inferior ao tamanho máximo do segmento de dados TCP, e não se o seu tamanho corrente for inferior a 25% do seu tamanho original, como sugerido por CLARK. Desta forma, o tempo de espera por parte dos usuários é menor e os resultados são os mesmos, já que o intuito é evitar a transmissão de mensagens pequenas.

Já na recepção de dados, foi utilizado o esquema proposto por CLARK [25] de enviar "acks" apenas quando a janela estiver quase completa ou no caso da chegada de mensagens com o comando PUSH [17], reduzindo o número de

"acks" enviados e contribuindo, assim, para o desempenho do sistema de comunicações remoto, que trata um número menor de mensagens. Este esquema gera, ainda, uma diminuição artificial na janela, pois enquanto os "acks" não chegam as mensagens da janela seguinte não podem ser enviadas, contribuindo, assim, para a redução da ocorrência da síndrome como é demonstrado no mesmo artigo. Desta forma, este esquema, além de evitar o envio de mensagens redundantes, evita também a síndrome.

IV.3.4. A Manipulação de Mensagens Urgentes

As mensagens urgentes [17] definidas pelo protocolo TCP são mensagens que têm prioridade sobre as outras e que, embora estejam incluídas no mesmo universo de números de seqüência e, por conseguinte, tenham seu envio sujeito à mesma janela que os dados normais, não obedecem à ordem cronológica imposta ao envio das mensagens. Os dados de uma mensagem urgente podem ser enviados sozinhos ou compartilhar uma mensagem com dados de uma mensagem normal.

No sistema de comunicações do PLURIX, o tratamento do envio das mensagens urgentes é semelhante ao do envio das mensagens normais; a única diferença é que, por ter prioridade, as mensagens urgentes são inseridas na frente da fila de saída do "endpoint", depois de outras mensagens urgentes. Aliás, a prioridade das mensagens urgentes é a

causa pela qual a atribuição dos números de seqüência às mensagens só é feita no momento em que estas são enviadas efetivamente.

Já na recepção, embora os dados urgentes sejam copiados para a área do usuário antes de dados normais com números de seqüência inferiores aos deles, as mensagens que os contêm devem permanecer na fila de entrada para que os possíveis dados normais contidos na mesma mensagem sejam também copiados e para possibilitar o controle dos números de seqüência, vital para o êxito da comunicação.

Para viabilizar este mecanismo, toda mensagem TCP recebida é inserida na fila de entrada do "endpoint", de acordo com o seu número de seqüência. Caso uma mensagem inserida possua dados urgentes, um contador é incrementado. Assim, toda vez que o usuário requisita dados de um "endpoint", caso haja dados urgentes, o que é imediatamente verificado através do contador, a fila é percorrida de trás para frente a sua procura. Mesmo quando todos os dados urgentes de uma mensagem já foram copiados para a área do usuário, esta mensagem permanece na fila, perdendo apenas sua característica urgente. Caso não haja dados urgentes, a cópia dos dados normais será iniciada pela mensagem do fim da fila. Esta cópia vai passando de mensagem em mensagem enquanto houver espaço na área estabelecida pelo usuário.

IV.3.5. A Sincronização

Como foi mostrado, na implementação do sistema de comunicações do PLURIX, a realização de uma comunicação se dá pela cooperação entre o "daemon" e cada um dos processos de usuário que tenciona utilizar os serviços de comunicação. Esta cooperação se dá através do compartilhamento de estruturas de dados. Para manter a integridade destas estruturas, bem como sincronizar a execução dos diversos processos que as manipulam, são utilizados semáforos. Os semáforos possibilitam a exclusão mútua na manipulação de áreas de memória consideradas regiões críticas. Sua utilização, no núcleo do PLURIX, é feita através de pares de primitivas responsáveis pela sua alocação e liberação [3-26]. Dois pares de primitivas são utilizados no sistema de comunicações: O par "SPINLOCK/SPINFREE" e o par "SLEEPLOCK/SLEEPFREE".

A primitiva "SPINLOCK" mantém a UCP em "loop" até que o recurso esperado seja liberado pela "SPINFREE", alocando-o a seguir. Os semáforos manipulados por este par de primitivas devem ser usados para recursos que fiquem ocupados por um curto espaço de tempo. No sistema de comunicações, estes semáforos são utilizados para garantir a exclusão mútua no acesso a cada um dos "endpoints", ao "pool de buffers" e à fila de retransmissão do sistema.

A primitiva "SLEEPLOCK" força uma troca de contexto quando o recurso requerido está ocupado. Os semáforos

manipulados pelo par "SLEEPLOCK/SLEEPFREE" devem, portanto, ser usados para recursos que fiquem ocupados por um espaço de tempo maior que o gasto em uma troca de contexto. Um semáforo deste tipo é utilizado na manipulação da linha de comunicações, evitando, assim, que as mensagens enviadas sejam embaralhadas.

Um outro par de primitivas existente no PLURIX e utilizado no sistema de comunicações é o "EVENTWAIT/EVENTDONE" [3-26]. Estas primitivas são utilizadas para sincronizar processos que estejam ligados à ocorrência de um mesmo evento, ou seja, um processo depende da ocorrência de um evento gerado pelo outro. A primitiva "EVENTWAIT" força uma troca de contexto colocando o processo dependente a espera do evento cuja ocorrência será sinalizada pelo processo gerador através da primitiva "EVENTDONE".

No sistema de comunicações, os processos de usuário são os processos dependentes, enquanto que o "daemon" é o processo gerador dos eventos. Estes eventos estão diretamente relacionados aos "endpoints". Sua ocorrência indica, assim, a chegada de um pedido de conexão, a chegada de uma aceitação para um pedido de conexão previamente enviado, a chegada de dados ou a liberação do envio de dados (através de um espaço na janela ou na fila de saída). Outros eventos foram definidos pela XTI, como é mostrado na seção IV.1. No entanto, há um esquema de espera definido somente para as rotinas que manipulam o

T_LISTEN, o T_CONNECT, o T_DATA e o T_GODATA. A rotina t_rcvdis, por exemplo, não espera a ocorrência do evento T_DISCONNECT; ela retorna um erro, caso este não tenha ocorrido.

Quanto à manipulação dos eventos, foi criada, em cada "endpoint", uma fila que os armazena desde sua ocorrência até que a rotina apropriada [22] os retire da fila. À medida que os eventos vão ocorrendo, eles vão sendo colocados, pelo "daemon", na fila da qual serão retirados, pelos processos de usuário, de acordo com a ordem de chegada. Três rotinas são responsáveis pela manipulação desta fila: a eventverif (que verifica a ocorrência de determinado evento), a eventin (que coloca um evento na fila) e a eventout (que retira da fila seu evento mais antigo). O evento T_DISCONNECT, porém, quando ocorre, gera a retirada do todos os eventos da fila. Isto se deve ao fato de que sua prioridade é absoluta.

A XTI não prevê a indicação da ocorrência de eventos através de interrupção do processo do usuário e conseqüente desvio para uma rotina do próprio usuário. Pela sua definição, o usuário que utilizar o modo assíncrono deve monitorar a ocorrência de eventos através da rotina t_look, que utiliza a rotina eventverif para verificar a ocorrência de eventos. Já o esquema utilizado pelo sistema de Berkeley permite que o usuário espere sincronamente por vários eventos através da rotina select [24]. No PLURIX, foi seguido o padrão definido pela XTI.

Assim, atualmente, um programa de usuário executando em modo assíncrono deve fazer um "loop de status" para perceber a ocorrência de um evento. A intenção é, no entanto, criar um esquema que torne possível ao usuário definir uma rotina de interrupção a ser ativada no momento da ocorrência de qualquer evento. A implementação deste esquema utilizará o mecanismo de sinais já disponível no PLURIX [4].

IV.3.6. As Incompatibilidades com as "system calls"

O fato de se ter utilizado os "drivers", para o acesso ao núcleo do sistema operacional, implicou em algumas dificuldades na implementação de certas rotinas da XTI. A maior parte destas dificuldades ocorreu por problemas de incompatibilidade entre os parâmetros definidos para as rotinas da XTI e os parâmetros estabelecidos para as rotinas do sistema utilizadas na entrada do mesmo. Por causa destes problemas, foram criadas estruturas contendo os parâmetros necessários, sendo a entrada no sistema resolvida pela chamada à rotina ioctl, que recebe e passa para a ioctl do "driver" um ponteiro para uma área de memória. Este ponteiro pode ser o endereço de uma estrutura, resolvendo, assim, o problema. Neste caso se enquadram as rotinas de leitura (t_rcv) e escrita (t_snd) do "connection mode", que além do descritor do arquivo (ou "endpoint"), da área de memória e da quantidade de dados, utilizam um "flag"

indicativo de mensagem urgente e da existência de mais dados.

Como foi mostrado anteriormente, a especificação de um "endpoint" nas chamadas às rotinas da interface é feita pelo seu descritor. Desta forma, todas as rotinas da interface recebem e passam ao sistema (pelas "system calls") o descritor do "endpoint" especificado. A rotina t_accept é uma exceção; seu modo de funcionamento deve permitir a especificação de dois "endpoints". Um deles é o "endpoint" para o qual chegou um pedido de conexão, e o outro é o que o usuário vai utilizar para responder o pedido e para a manutenção da comunicação. Isto significa, como foi mostrado, que um usuário pode receber o pedido de conexão por um "endpoint", e concretizar esta conexão em outro. Para resolver esta incoerência com o resto do sistema foram criadas dentro do sistema três rotinas accept (accept1, accept2 e accept3), que são ativadas apropriadamente pela rotina t_accept. A accept1 é utilizada no caso mais simples, em que os dois descritores são iguais, ou seja, ela envia a aceitação pelo mesmo "endpoint" que recebeu o pedido de conexão. A accept2 não executa propriamente uma função de accept; ela simplesmente marca com o identificador do processo o "endpoint" que recebeu o pedido de conexão. Já a accept3 estabelece a conexão em um novo "endpoint"; para isso ela procura na lista de "endpoints" o que foi marcado pela accept2 com o identificador do processo, pois este "endpoint" possui os dados da conexão a se formar. Assim,

foi utilizado o identificador do processo para tornar possível o relacionamento entre dois "endpoints" especificados pelo usuário em uma mesma entrada no sistema.

CAPÍTULO V

TESTES E IMPLANTAÇÃO

V.1. Testes Realizados ao Longo da Implementação

A implementação do sistema de comunicações foi dividida em etapas e cada etapa teve sua fase de testes. A primeira etapa foi a implementação da parte básica do sistema de comunicações. Assim, foi definida a estrutura do "daemon", bem como a organização do funcionamento dos protocolos. Foram implementadas as rotinas referentes ao tratamento do protocolo IP (em uma versão simplificada, enviando todas as mensagens para o próprio sistema através do modo interno), e a parte local da interface de transporte contendo a criação e o término da utilização dos "endpoints", e a associação e desassociação destes aos "ports". Nesta fase da implementação, foi criada a estrutura dos "endpoints". Foi decidido, assim, que haveria uma lista de "endpoints" para cada protocolo (UDP e TCP, por exemplo), e que cada "endpoint" seria uma estrutura contendo todas as informações necessárias à comunicação.

A segunda etapa foi a implementação do modo de comunicação sem conexão ("connectionless mode"). Assim, foram implementados o protocolo UDP e as rotinas da

interface de transporte correspondentes ao "connectionless mode" (transmissão de datagramas). Nesta etapa, os testes do sistema foram realizados através da utilização do modo interno de comunicação, já que o intuito era testar o funcionamento da interface e dos protocolos, e não a transmissão de mensagens.

Para testar o "connectionless mode", foram construídos um servidor de "echo", cuja função é retransmitir as mensagens recebidas para os seus respectivos remetentes, e clientes para enviar mensagens ao servidor e recebê-las de volta verificando sua correção.

Em uma terceira etapa, o sistema de comunicações foi ampliado, passando a realizar também transmissões via linha serial, através da utilização do protocolo SLIP [13]. Nesta fase foi possível a transmissão de datagramas para outro computador.

A quarta etapa foi a implementação do protocolo TCP e das rotinas relativas à conexão, à desconexão e à transmissão de mensagens em circuitos virtuais. Inicialmente, foi desenvolvido um subconjunto com funções restritas, que foi ampliado paulatinamente. O "connection mode" foi depurado com a utilização de uma linha serial ligando o computador nele mesmo. Desta forma foi simulado um ambiente real, sem ser necessária a instalação do núcleo do sistema em outro computador a cada alteração do

sistema.

Para os testes do "connection mode", foram criados um servidor e um cliente. Inicialmente, o servidor aceitava apenas um pedido de conexão, após o qual recebia e colocava na saída padrão o arquivo a ele enviado pelo cliente. Posteriormente, o servidor foi estendido para esperar por pedidos de conexão e criar, para cada um, um novo processo servidor para aceitar o pedido e concretizar a conexão. Outra extensão implementada foi o envio, pelo cliente, em mensagem urgente, do nome do arquivo a ser enviado. O servidor, por sua vez, passou a abrir um arquivo, com o nome enviado pelo cliente, para colocar os dados recebidos.

V.2. A Implantação do Sistema de Comunicações

A existência de um sistema de comunicações básico no PLURIX não é suficiente para que os seus usuários tenham acesso aos serviços disponíveis nas redes. Isto porque o acesso a estes serviços é realizado pelos programas de aplicação, que não fazem parte do sistema de comunicações básico.

Por não haver disponibilidade de programas de aplicação que utilizassem a interface XTI, uma solução paliativa para a utilização imediata do sistema de comunicações do PLURIX seria a instalação de programas de

aplicação desenvolvidos para utilizar os "sockets" de um sistema UNIX-like. No entanto, para que programas de aplicação desenvolvidos para utilizar os "sockets" pudessem ser instalados no PLURIX, seria necessário criar uma série de rotinas conversoras do padrão "socket" para o padrão XTI. Assim, uma quinta etapa de implementação foi necessária à implantação do sistema de comunicações do PLURIX. Nesta etapa foram criadas as rotinas conversoras que viabilizariam a utilização imediata do sistema de comunicações.

A criação das rotinas conversoras deveria ser uma tarefa simples, já que, de modo geral, as rotinas definidas para a utilização dos "sockets" e as rotinas definidas pela XTI possuem funções similares, utilizando apenas parâmetros diferentes. No entanto, a existência de algumas características particulares dos "sockets", bem como a ausência de algumas funções na definição da XTI, exigiram a implementação de algumas extensões no sistema de comunicações.

Um exemplo de função ausente na definição da XTI é a realizada pela rotina select [24]. Foi necessário, por isso, emular o seu funcionamento através da criação de uma rotina que espera pela ocorrência de um evento (indicado por um sinal) por um determinado intervalo de tempo.

Uma incompatibilidade encontrada entre os "sockets" e a XTI foi a diferença de funcionamento existente entre as

rotinas listen e accept dos "sockets" e t_listen e t_accept da XTI. A listen apenas indica ao sistema que o "socket" especificado está esperando um pedido de conexão, enquanto a accept recebe o pedido e o responde através de outro "socket" criado por ela mesma. Já na XTI, a t_listen recebe os pedidos de conexão que são respondidos pela t_accept pelo mesmo "endpoint", ou por outro determinado pelo usuário, que é o responsável pela sua criação. A solução utilizada para contornar a diferença foi criar uma rotina accept conversora que espera e recebe um pedido de conexão (chamando a t_listen), criando, a seguir, um "endpoint" (chamando a t_open) para respondê-lo (chamando a t_accept).

Outra incompatibilidade encontrada foi gerada pelo fato de os "sockets" considerarem também como snd e rcv as "system calls" write e read. Na implementação do sistema de comunicações do PLURIX, como foi mostrado na seção IV.3.6, as rotinas t_snd e t_rcv definidas pela XTI utilizam para entrar em modo supervisor a "system call" ioctl e não as "system calls" write e read, por razões de incompatibilidade entre seus parâmetros. Assim, foi necessário criar, dentro do "driver" implementado, rotinas alternativas para enviar e receber mensagens, que pudessem ser iniciadas por uma chamada à rotina write ou à rotina read, respectivamente.

Uma característica particular dos "sockets" é permitir que o usuário determine que as mensagens urgentes

sejam recebidas na ordem estabelecida pelo seu número de seqüência, e não antes de dados normais com número de seqüência menor (opção OOBINLINE - "out of band in line" [24]). Como a XTI não define tal funcionamento, esta opção foi incluída entre as opções manipuladas pela rotina t_optmgmt (definida pela XTI [22]) e, quando especificada, faz com que mensagens urgentes sejam tratadas como se fossem mensagens normais.

A vantagem de se resolver as diferenças entre as interfaces através de rotinas conversoras é o fato de se poder usar estas rotinas para a instalação de quaisquer programas de aplicação que utilizem os "sockets". Utilizando as rotinas conversoras, a instalação destes programas será praticamente imediata, não exigindo, a cada instalação, um novo trabalho de adaptação.

CAPÍTULO VI

CONCLUSÕES E EXTENSÕES

Em uma avaliação geral do sistema de comunicações implementado, podemos dizer que foi atingido o objetivo de dotar o PLURIX de um mecanismo capaz de viabilizar a sua comunicação com outros sistemas. Este mecanismo básico foi construído de forma a permitir expansões visando um aumento na sua funcionalidade e uma melhora no seu desempenho.

A decisão de implementar os protocolos TCP/IP e de conseqüentemente não seguir o modelo definido pela ISO (International Standards Organization) se mostrou bastante adequada, já que é grande, ainda, o número de sistemas utilizando a família TCP/IP. No entanto, a tendência em direção ao padrão ISO é um fato, e, por esta razão, foi importante a escolha, para padrão da interface de transporte, da XTI em detrimento dos "sockets". A interface XTI foi definida voltada para o padrão ISO, e por isso sua presença no sistema de comunicações do PLURIX favorecerá uma futura migração para este padrão. Por outro lado, por serem os "sockets" utilizados na maior parte dos aplicativos disponíveis, sua rejeição implicou na necessidade de se implementar rotinas conversoras entre as utilizadas nos "sockets" e as definidas pela XTI, como foi

mostrado no capítulo V.

No que diz respeito à implementação da interface XTI, a utilização dos "drivers" foi decisiva para a sua perfeita ambientação no sistema operacional. O fato de ser evitada a criação de numerosas "system calls" compensou as poucas incompatibilidades encontradas.

A utilização da linha serial para a comunicação, embora essencial para a validação e depuração do sistema, não se mostrou eficiente. Sua baixa velocidade determina um grande atraso na comunicação, como foi constatado através da utilização do modo interno, com o qual o sistema apresentou um desempenho extremamente superior. A deficiência da comunicação serial pode ser facilmente superada pela utilização de uma interface Ethernet [12]. Atualmente, o computador EBC 32010, onde está sendo desenvolvido o sistema de comunicações, não possui tal interface, estando prevista para breve a sua instalação.

A utilização de uma interface Ethernet implicará em uma extensão no algoritmo de roteamento realizado no nível de interconexão. Além disso, será necessário implementar outros protocolos, como o ARP (Address Resolution Protocol) [27], responsável pela transformação dos endereços "internet" em Ethernet.

A eficiência de um sistema de comunicações se reflete não só no seu desempenho, mas também na utilização

apropriada da rede. O bom desempenho do sistema foi alcançado essencialmente através da utilização de artifícios para não só minimizar, mas também agilizar, as tarefas realizadas pelo "daemon", ponto central do sistema de comunicações. Entre os principais artifícios empregados, podemos citar:

- o compartilhamento de um único "buffer" entre os protocolos dos diversos níveis, evitando a transferência excessiva e dispendiosa dos dados.
- a atribuição da tarefa de enviar mensagens aos processos dos usuários.
- a atribuição da tarefa de manipular os dados, ou seja, de copiá-los do modo usuário para o modo supervisor e vice-versa, unicamente aos processos dos usuários.
- a organização das filas de mensagens dos "endpoints" de forma a delegar aos processos dos usuários a parte mais dispendiosa da sua manipulação.

A utilização apropriada da rede, caracterizada principalmente pelo não envio de mensagens redundantes, foi garantida pela manipulação adequada da janela e do envio de "acks" no protocolo TCP. Esta manipulação adequada resultou principalmente do não envio de "acks" a cada mensagem recebida, o que contribuiu para a redução das mensagens em circulação e para o impedimento da ocorrência da síndrome da janela pequena, pela diminuição artificial da janela. Outro fator definitivo para o impedimento da ocorrência da síndrome da janela pequena foi a decisão de não moldar as mensagens a serem enviadas

de acordo com o tamanho corrente da janela, mas colocar em cada mensagem o número máximo possível de dados do usuário, limitando este número apenas pelo tamanho máximo do segmento de dados estabelecido pelo destino.

Como foi mostrado ao longo do texto, alguns mecanismos utilizados na implementação do sistema de comunicações não se mostraram totalmente satisfatórios, ficando seu aperfeiçoamento reservado para futuras extensões do sistema. Um exemplo destes mecanismos é a busca seqüencial realizada na fila de "endpoints" no momento da chegada de uma mensagem. Por ser realizada pelo "daemon", a agilização desta busca, pela utilização de um "cache" ou de algoritmos de "hash", resultará em uma melhoria no desempenho do sistema.

Outras extensões previstas para o sistema são:

- a inclusão dos "raw-endpoints", que tornarão possível o acesso dos programas de usuários aos protocolos do nível de interconexão, como o IP.
- a criação de um esquema assíncrono que permita ao sistema de comunicações informar o usuário sobre a ocorrência de eventos.

Como a existência de um sistema de comunicações básico no PLURIX não é suficiente para que os seus usuários tenham acesso aos serviços disponíveis nas redes, visto que o acesso a estes serviços é realizado pelos programas de aplicação, que não fazem parte do sistema de

comunicações, sugerimos para futuros trabalhos o desenvolvimento de programas que executem os protocolos de aplicação. Estes programas, através da utilização do sistema de comunicações, realizarão a pretendida integração entre o PLURIX e o mundo externo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] FALLER, N. et alii, "O Projeto PEGASUS-32X/PLURIX", Anais do XVII Congresso Nacional de Informática, Rio de Janeiro, Novembro 1984.

[2] FALLER, N., ANIDO, M. L. e SALENBAUCH, P., "Técnicas de Projeto Utilizadas na Construção do Supermicrocomputador PEGASUS-32X e do Sistema Operacional PLURIX", Anales de La Convención Informática Latina, CIL 85, Barcelona, Abril 1985.

[3] FALLER, N. and SALENBAUCH, P., "PLURIX: A multiprocessing UNIX-like Operating System", Proceedings of the Second IEEE Workshop on Workstation Operating System, pp. 29-36, Washington, September 1989.

[4] FALLER N. et alii, "Manual de Referência PLURIX 2.1", Rio de Janeiro, NCE/UFRJ, 2a. Edição, Junho 1989.

[5] BACH, M. J., "The Design of the UNIX Operating System", New Jersey, Prentice Hall, 1st Edition, 1986.

[6] COMER, D., "Internetworking with TCP/IP - Principles, Protocols and Architecture", New

Jersey, Prentice Hall, 1st Edition, 1988.

[7] WATSON R. W. and MAMRAK S. A., "Gaining Efficiency in Transport Services by Appropriate Design and Implementation Choices", ACM Transactions on Computer Systems. 5(2): 97-120, May 1987.

[8] HUTCHINSON, N. C. and PETERSON, L. L., "Design of the x-Kernel", ACM Computer Communications Review. 18(4): 65-75, 1988.

[9] RASHID, R. F. and ROBERTSON, G. G., "Accent: A communication oriented network operating system kernel", ACM Operating Systems Review. 15(5): 64-75, December 1981.

[10] CHERITON, D. R. and ZWAENEPOEL, W., "Distributed Process Groups in the V Kernel", ACM Transactions on Computer Systems. 3(2): 77-107, May 1985.

[11] COMER, D., "Operating System Design (Volume II) - Internetworking with XINU", New Jersey, Prentice Hall, 1st Edition, 1987.

[12] RIBEIRO FILHO, J. L., "O padrão Ethernet para redes locais", Boletim do PLURIX, Rio de Janeiro, NCE/UFRJ, 3(13): 3-6, Ago/Set/Out 1990.

[13] ROMKEY, J., "A nonstandard for transmission of IP

Datagrams over serial lines: SLIP", Request For Comments 1055, Network Working Group, June 1988.

[14] TANENBAUM, A. S., "Computer Networks", New Jersey, Prentice Hall, 2nd Edition, 1988.

[15] POSTEL, J., "File Transfer Protocol", Request For Comments 959, DDN Protocol Handbook, Volume 2, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

[16] POSTEL, J., "TELNET", Request For Comments 854, DDN Protocol Handbook, Volume 2, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

[17] POSTEL, J., "Transmission Control Protocol", Request For Comments 793, DDN Protocol Handbook, Volume 2, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

[18] POSTEL, J., "User Datagram Protocol", Request For Comments 768, DDN Protocol Handbook, Volume 2, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

[19] POSTEL, J., "Internet Protocol", Request For Comments 791, DDN Protocol Handbook, Volume 2, DDN

Network Information Center, SRI International, Menlo Park, California, December 1985.

[20] AT&T, "UNIX System V: Network Programmer's Guide", New Jersey, Prentice Hall, 1st Edition, 1987.

[21] MANNAS, J., "Two Views One Vision", UNIX Review, 6(10): 77-83, October 1988.

[22] X/OPEN Company, Ltd., "X/OPEN Portability Guide: Networking Services", New Jersey, Prentice Hall, 1st Edition, 1988.

[23] CLARK, D. D., "Modularity and Efficiency in Protocol Implementation", Request For Comments 817, DDN Protocol Handbook, Volume 3, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

[24] 4.3 BERKELEY SOFTWARE DISTRIBUTION, "UNIX Programmer's Manual", U.S.A, Berkeley University, 7th Edition, 1986.

[25] CLARK, D. D., "Window and Acknowledgement Strategy in TCP", Request For Comments 813, DDN Protocol Handbook, Volume 3, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

[26] FALLER, N. e SALENBAUCH, P., "PLURIX, O Sistema Operacional Multiprocessador do NCE-UFRJ: (1) Sincronização de Processos", Data News, 24 de Setembro de 1985.

[27] PLUMMER, D. C., "An Ethernet Address Resolution Protocol", Request For Comments 826, DDN Protocol Handbook, Volume 3, DDN Network Information Center, SRI International, Menlo Park, California, December 1985.

APÊNDICE

EXEMPLO DE UTILIZAÇÃO DO SISTEMA DE COMUNICAÇÕES

Neste apêndice será mostrada a seqüência de procedimentos necessários à utilização do "connection mode" do sistema de comunicações. A figura 1 relaciona os procedimentos executados por um cliente e um servidor que se comunicam. A explicação acerca dos procedimentos é baseada no exemplo cujos programas são apresentados no final deste apêndice. O exemplo consta de um cliente, que envia um arquivo a um servidor, e de um servidor, que recebe um arquivo e o coloca na saída padrão.

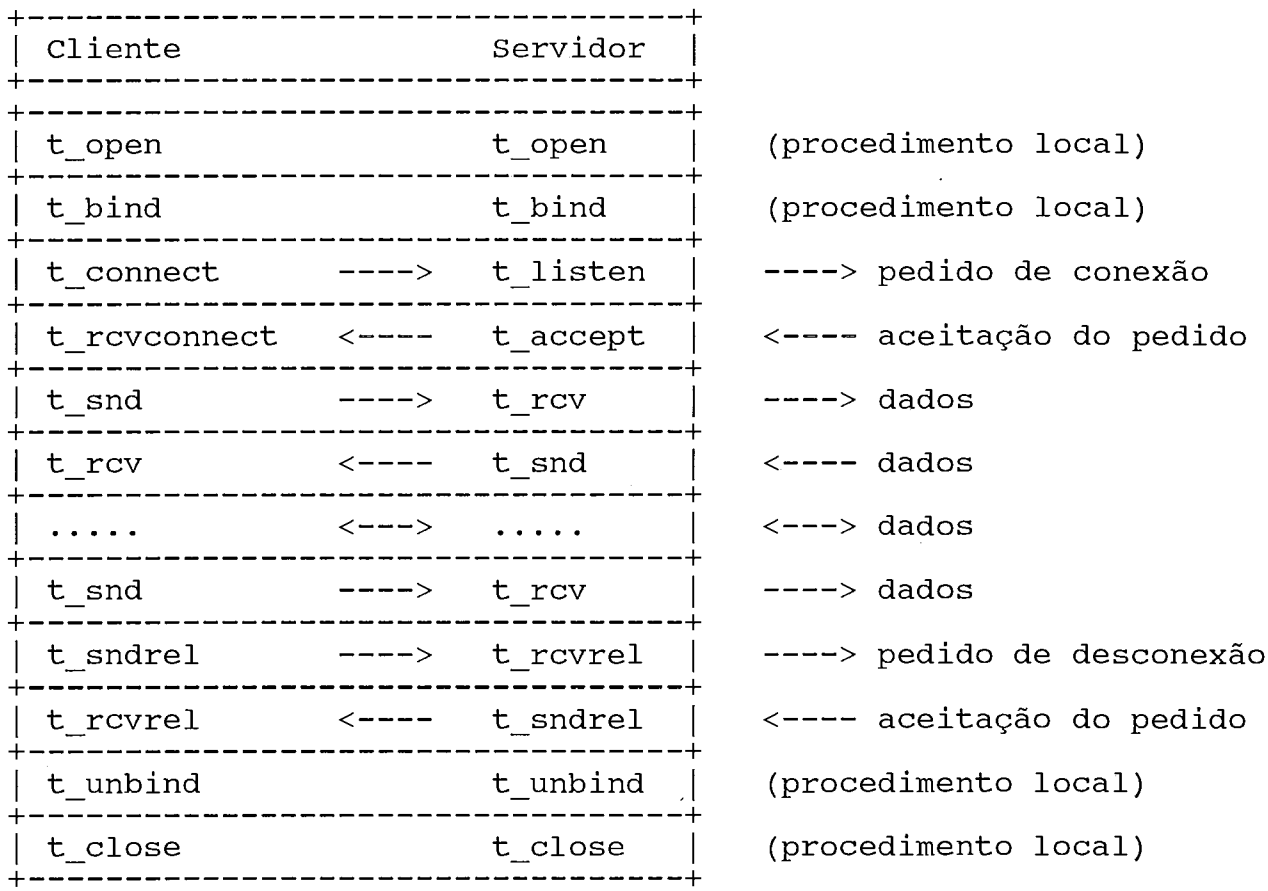

Figura 1

Seqüência de procedimentos necessários
à utilização do "connection mode".

O servidor, inicialmente, chama a rotina t_open para criar um "endpoint". O parâmetro "/dev/itntcp", utilizado no programa exemplo, indica a utilização do protocolo TCP ("connection mode") na comunicação. O valor retornado pela t_open é o descritor do "endpoint" criado e deve ser utilizado sempre que se desejar referenciá-lo. Depois de criar o "endpoint", o servidor deve associá-lo a um endereço ("port") previamente estabelecido ("well known"), para que os programas clientes possam iniciar a

comunicação com ele. Para isto é utilizada a rotina t_bind, que no exemplo associa o "endpoint" ao "port" 21 (veja a descrição das estruturas na listagem que antecede o programa exemplo). Como é visto no exemplo, o servidor deve testar se o "port" associado ao "endpoint" foi mesmo o 21, já que o sistema pode associar outro, se o escolhido estiver ocupado. Depois deste procedimento local, o servidor se encontra pronto para receber pedidos de conexão. Estes pedidos de conexão, que podem ser mandados por diversos clientes, são recebidos pela rotina t_listen que, no modo síncrono, fica à espera deles.

O cliente, inicialmente, executa o mesmo procedimento que o servidor, criando um "endpoint" e o associando a um "port". Porém, ao contrário do que ocorre com o servidor, o "port" não precisa ser previamente estabelecido e o cliente pode deixar esta escolha a cargo do sistema. No exemplo, isto é mostrado pelos argumentos nulos da chamada à rotina t_bind. Após a fase inicial, o cliente inicia a conexão através da chamada à rotina t_connect, que envia um pedido de conexão ao servidor especificado pelo "port" e endereço na rede. No exemplo, o cliente pede uma conexão ao servidor cujo "port" é o 21 e o endereço na rede é zero, indicando a utilização do modo interno.

Quando a mensagem contendo o pedido de conexão chega ao "endpoint" destino, a rotina t_listen retorna ao servidor, indicando-lhe o "port" e o endereço na rede do remetente do pedido. No exemplo estes dados estarão na

estrutura callb. Para aceitar o pedido, o servidor chama a rotina t_accept. No exemplo, o servidor aceita o pedido através do mesmo "endpoint" que o recebeu, por isso os dois primeiros parâmetros utilizados na chamada à t_accept especificam o mesmo "endpoint". A rotina t_accept envia, assim, a aceitação do pedido de conexão ao cliente.

Quando a aceitação do pedido de conexão é recebida pelo "endpoint", o cliente pode recebê-la através da rotina t_rcvconnect, que retorna, em uma estrutura callb, o "port" e o endereço na rede do servidor. Outra opção, utilizada pelo exemplo, é a espera pela aceitação na própria t_connect (modo síncrono). Neste caso, o terceiro argumento da t_connect, se especificado, será um ponteiro para uma estrutura callb que conterá o "port" e o endereço na rede do servidor. Depois de recebida a aceitação, a conexão está formada e o cliente e o servidor podem iniciar a transmissão de dados. A rotina t_snd é utilizada para enviar dados, enquanto a t_rcv é utilizada para recebê-los. No exemplo, apenas o cliente envia dados; ele lê o arquivo a ser enviado e o transmite através da rotina t_snd. O servidor, por sua vez, recebe os dados e os coloca na saída padrão.

A rotina t_sndrel é utilizada para enviar o pedido de término da conexão, enquanto a t_rcvrel é utilizada para receber este pedido. Cada um, cliente e servidor, deve enviar um pedido de desconexão e receber o enviado pelo outro. No exemplo, após enviar o arquivo, o cliente chama

a t_sndrel, que envia uma mensagem de pedido de desconexão. O servidor, ao detectar a chegada deste pedido (pelo erro retornado pela t_rcv), o recebe através da rotina t_rcvrel, chamando, a seguir, a t_sndrel para confirmar a desconexão. A t_sndrel envia então a aceitação do pedido de desconexão, que, ao chegar ao cliente, é recebido pela t_rcvrel, efetivando o fim da conexão.

Depois de desfeita a conexão, são utilizadas as rotinas t_unbind, que desassocia um "endpoint" do "port" associado anteriormente, e a t_close, que libera o "endpoint" alocado. No entanto, no exemplo, somente o cliente chama estas rotinas. O servidor volta a chamar a rotina t_listen para esperar novos pedidos de conexão; sua execução é contínua, só sendo interrompida por um sinal ou por um erro.

A utilização do protocolo UDP ("connectionless mode") é semelhante à do TCP. A diferença básica é que, por não haver formação de conexão, cada usuário deve especificar, junto com os dados a serem enviados, o "port", que no caso do servidor deve ser previamente estabelecido, e o endereço na rede do usuário remoto. Na recepção, a interface também fornecerá estes dados para o usuário juntamente com cada mensagem recebida.

```
/*
 ****************************************************************
 *      Constantes e estruturas (definidas pela XTI)            *
 *      utilizadas nos programas do exemplo.                    *
 ****************************************************************
 */

/*
 *      Eventos.
 */
#define T_LISTEN        0x0001  /* pedido de conexão */
#define T_CONNECT       0x0002  /* confirmação de conexão */
#define T_DATA          0x0004  /* dados */
#define T_EXDATA        0x0008  /* dados urgentes */
#define T_DISCONNECT    0x0010  /* pedido de desconexão */
#define T_UDERR         0x0040  /* erro em datagrama */
#define T_ORDREL        0x0080  /* desconexão ordenada */
#define T_GODATA        0x0100  /* fluxo de dados liberado */
#define T_GOEXDATA      0x0200  /* fluxo de dados urg liberado */

/*
 *      Formato das informações contidas nas estruturas.
 */
typedef struct  netbuf  NETBUF;

struct  netbuf
{
        ulong   maxlen;         /* tamanho de buf */
        ulong   len;            /* número de bytes em buf */
        char    *buf;           /* dados */

};

/*
 *      Estrutura utilizada pela rotina t_bind.
 */
typedef struct  t_bind  T_BIND;

struct  t_bind
{
        NETBUF  addr;   /* endereço ("port") */
        ulong   qlen;   /* tam da fila de pedidos de conexão */

};

/*
 *      Estrutura utilizada pelas rotinas t_connect,
 *      t_rcvconnect, t_listen e t_accept.
 */
typedef struct  t_call  T_CALL;

struct  t_call
{
        NETBUF  addr;           /* endereço */
        NETBUF  opt;            /* opções */
        NETBUF  udata;          /* mensagem */
        int     sequence;       /* índice da fila de
                                        pedidos de conexão */
```

```
};

/*
 *      Formato do endereço.
 */
typedef struct  addr    ADDR;

struct  addr
{
        unsigned long   a_port;         /* "port" */
        unsigned long   a_addr;         /* endereço na rede */

};
```

```
/*
 ****************************************************************
 *                                                              *
 *      Programa SERVIDOR:                                      *
 *                                                              *
 *      Recebe um arquivo através do "endpoint" associado       *
 *      ao "port" 21 e o coloca na saída padrão.                *
 *                                                              *
 ****************************************************************
 */
main (argc, argv)
int     argc;
char    *argv[];
{
        register        ret;
        int             ed;
        ADDR            addrloc;
        ADDR            addrrem;
        T_BIND          bindb;
        T_CALL          callb;
        buf             char[BUFSIZ];

        /*
         *      Cria um "endpoint" para utilizar o protocolo TCP.
         */
        if ((ed = t_open ("/dev/itntcp", O_RDWR, NULL)) < 0)
        {
                t_error (argv[0]);
                exit (1);
        }

        /*
         *      Associa o "endpoint" ao "port" 21,
         *      depois de preencher a estrutura bindb
         *      que também será usada para receber
         *      o "port" efetivamente associado ao "endpoint".
         */
        addrloc.a_port = 21;
        addrloc.a_addr = 0;
        bindb.qlen = 1;
        bindb.addr.maxlen = sizeof (ADDR);
        bindb.addr.len = sizeof (ADDR);
        bindb.addr.buf = (char *)&addrloc;

        if (t_bind (ed, &bindb, &bindb) < 0)
        {
                t_error (argv[0]);
                t_close (ed);
                exit (1);
        }

        /*
         *      Apenas o "port" 21 é aceito.
         */
        if (addrloc.a_port != 21)
        {
                t_error (argv[0]);
                t_unbind (ed);
```

```
        t_close (ed);
        exit (1);
}

for (;;)
{
        /*
         *      Espera por pedidos de conexão.
         *      O remetente do pedido será colocado
         *      na estrutura callb.
         */
        callb.addr.maxlen = sizeof (ADDR);
        callb.addr.buf = (char *)&addrrem;
        callb.opt.len = 0;
        callb.udata.len = 0;

        if (t_listen (ed, &callb) < 0)
        {
                t_error (argv[0]);
                t_unbind (ed);
                t_close (ed);
                exit (1);
        }

        /*
         *      Aceita um pedido de conexão recebido,
         *      cujo remetente está na estrutura callb.
         */
        if (t_accept (ed, ed, &callb) < 0)
        {
                t_error (argv[0]);
                t_unbind (ed);
                t_close (ed);
                exit (1);
        }

        /*
         *      Recebe os caracteres enviados
         *      pelo cliente e coloca-os na saída padrão.
         */
        while ((ret = t_rcv (ed, buf, BUFSIZ, 0)) > 0)
        {
                for (i = 0; i < ret; i++)
                        putchar ((int)buf[i]);
        }

        /*
         *      Recebe o pedido de fim de conexão.
         */
        if (t_rcvrel (ed) < 0)
        {
                if (t_look (ed) == T_DISCONNECT)
                {
                        /*
                         *      Aceita o pedido de
                         *      desconexão abrupta
                         *      recebido.
                         */
```

```
                        t_rcvdis (ed, NULL);
                        t_unbind (ed);
                        t_close (ed);
                        exit (1);
                }

                t_error (argv[0]);
                t_snddis (ed, NULL);
                t_unbind (ed);
                t_close (ed);
                exit (1);
        }

        /*
         *      Confirma o fim da conexão.
         */
        if (t_sndrel (ed) < 0)
        {
                t_error (argv[0]);
                t_unbind (ed);
                t_close (ed);
                exit (1);
        }
    }

}       /* fim do programa SERVIDOR */
```

```
/*
 ****************************************************************
 *                                                              *
 *      Programa CLIENTE:                                       *
 *                                                              *
 *      Envia uma arquivo ao servidor associado ao "port" 21.   *
 *                                                              *
 ****************************************************************
 */
main (argc, argv)
int     argc;
char    *argv[];
{
        int             ed;
        int             fd;
        ADDR            addrrem;
        T_CALL          callb;
        register        ret;
        char            buf[BUFSIZ];

        /*
         *      Abre o arquivo a ser enviado.
         */
        if ((fd = open (argv[1], O_RDONLY)) < 0)
        {
                t_error (argv[0]);
                exit (1);
        }

        /*
         *      Cria um "endpoint" para utilizar o protocolo TCP.
         */
        if ((ed = t_open ("/dev/itntcp", O_RDWR, NULL)) < 0)
        {
                t_error (argv[0]);
                exit (1);
        }

        /*
         *      Associa o "endpoint" a um "port" qualquer.
         */
        if (t_bind (ed, (T_BIND *)NULL, (T_BIND *)NULL) < 0)
        {
                t_error (argv[0]);
                t_unbind (ed);
                t_close (ed);
                close (fd);
                exit (1);
        }

        /*
         *      Envia um pedido de conexão ao servidor
         *      associado ao "port" 21 (depois de preencher a
         *      estrutura callb), e espera a aceitação.
         */
        addrrem.a_port = 21;
        addrrem.a_addr = 0;
        callb.addr.len = sizeof (ADDR);
```

```
callb.addr.buf = (char *)&addrrem;
callb.opt.len = 0;
callb.udata.len = 0;

if (t_connect (ed, &callb, (T_CALL *)NULL) < 0)
{
        t_error (argv[0]);
        t_unbind (ed);
        t_close (ed);
        close (fd);
        exit (1);
}

/*
 *      Lê os caracteres do arquivo
 *      e os envia ao servidor.
 */
while ((ret = read (fd, buf, BUFSIZ)) > 0)
{
        if (t_snd (ed, buf, ret, 0) < 0)
        {
                t_error (argv[0]);
                t_unbind (ed);
                t_close (ed);
                close (fd);
                exit (1);
        }

}

close (fd);

/*
 *      Envia um indicativo de término de conexão.
 */
if (t_sndrel (ed) != 0)
{
        t_error (argv[0]);
        t_unbind (ed);
        t_close (ed);
        exit (1);
}

/*
 *      Espera uma resposta do servidor.
 *      A rotina t_look retorna o evento ocorrido.
 */
do
{
        ret = t_look (ed);
}
while (ret != T_ORDREL  &&  ret != T_DISCONNECT);


if (ret == T_DISCONNECT)
{
        /*
         *      Aceita o pedido de desconexão
```

```
                 *      abrupta recebido.
                 */
                if (t_rcvdis (ed, NULL) != 0)
                        t_error (argv[0]);
                t_unbind (ed);
                t_close (ed);
                exit (1);
        }

        /*
         *      Recebe o indicativo de término de conexão.
         */
        if (t_rcvrel (ed) != 0)
        {
                t_error (argv[0]);
                t_unbind (ed);
                t_close (ed);
                exit (1);
        }

        t_unbind (ed);
        t_close (ed);

        exit (0);

}       /* fim do programa CLIENTE */
```