BRUNO BARBOSA ALBERT

IMPLEMENTAÇÃO DE UMA INTERFACE X.25

Dissertação apresentada ao Curso de MESTRADO em ENGENHARIA ELÉTRICA da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do Grau de Mestre.

AREA DE CONCENTRAÇÃO: PROCESSAMENTO DA INFORMAÇÃO

JOSÉ ANTÃO BELTRÃO MOURA
Orientador

JACQUES PHILIPPE SAUVÉ
Co-orientador

CAMPINA GRANDE
JUNHO - 1986

A333i Albert, Bruno Barbosa
Implementacao de uma interface X. 25 / Bruno Barbosa Albert. - Campina Grande, 1986.
89 f. : il.

Dissertacao (Mestrado em Engenharia Eletrica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Conversores - 2. Computacao - 3. Conversor X. 25 - 4. Procesamento de Dados 5. Dissertacao I. Moura, Jose Antao Beltrao, Prof. II. Sauve, Jacques Philippe, Prof. III. Universidade Federal da Paraiba - Campina Grande (PB) IV. Título

CDU 004.5(043)

IMPLEMENTAÇÃO DE UMA INTERFACE X.25

BRUNO BARBOSA ALBERT

DISSERTAÇÃO APROVADA EM

JOSÉ ANTÃO BELTRÃO MOURA
Orientador

JACQUES PHILLIPE SAUVÉ
Co-orientador

PAULO ROBERTO FREIRE CUNHA
Componente da Banca

JOBERTO SÉRGIO BARBOSA MARTINS
Componente da Banca

CAMPINA GRANDE - Pb
JUNHO - 1986

AGRADECIMENTOS

Gostaria de agradecer a todos que de alguma forma contribuiram para a execução deste trabalho. Em particular ao Senhor Eutacílio, a Dona Helena e a senhorita Alice pela confiança e pelo apoio que nunca faltaram.

RESUMO

O trabalho apresenta a implementação de uma interface entre um Conversor X.25 para a rede de comutação de pacotes CEPINNE e uma Estação de Transporte. Inicialmente, descreveremos detalhadamente os sinais de entrada e de saída da interface. Em seguida mostraremos como estes sinais interagem, utilizando um modelo de Máquina de Estados Finita para isso. A implementação será baseada no modelo definido. As dificuldades e as soluções encontradas durante o desenvolvimento do projeto serão amplamente discutidas. No final apresentaremos como foram realizados os testes.

SUMÁRIO

CAPÍTULO 1: INTRODUÇÃO

A necessidade de uma troca mais rápida e mais eficiente de informação entre usuários de diferentes sistemas levou a criação de redes de computadores. A técnica de comutação de pacote, é utilizada nessas redes, devido a natureza do tráfego gerado por um computador (alta intensidade mas de curta duração), o que permite que os canais de comunicação sejam compartilhados, implicando na diminuição dos custos de comunicação [SAUV 84].

As primeiras redes de comutação de pacotes surgiram na década de 60 para uma variedade de aplicações. Por exemplo, a ARPANET, desenvolvida inicialmente para fins militares nos EUA, a SITA para sistemas de reservas de passagens aéreas em escala mundial, a CYCLADES para pesquisas na França [COHE 78].

Em 1974, foram construídas redes públicas de comutação de pacotes, na França, Canadá, EUA, Reino Unido, Espanha e Japão, permitindo o acesso a pequenas firmas [SLOM 78].

A necessidade de comunicação além dos limites nacionais ou locais, implica que as regras que regulam a troca de informação, chamadas de protocolos, sejam padronizadas, de maneira a facilitar a conexão dos dois fins de comunicação [SAUV PAN]. Nesse sentido, foi introduzido, no final da década de 70 pela Organização Internacional para Padronização (ISO), um Modelo de Referência para a

Interconexão de Sistemas Abertos (RM/OSI).

O termo "Interconexão de Sistemas Abertos (OSI)" é utilizado de duas maneiras: para se referir ao esforço global no desenvolvimento de padrões na arquitetura RM/OSI; ou para qualificar padrões especificos para troca de informações entre sistemas que são abertos um para o outro, no sentido de que usam mutuamente padrões OSI aplicáveis [CHAP 83].

A arquitetura do RM/OSI se apresenta em sete camadas independentes uma da outra, como mostra a figura 1.1. As camadas são independentes no sentido de que qualquer modificação realizada numa determinada camada não afeta as outras camadas. A seguir daremos uma descrição sucinta das funções de cada camada:

1. Camada Física - provê as características elétricas e mecânicas para transmissão de bits num canal de comunicação.

2. Camada de Enlace - provê um enlace "virtual" sem erros e com controle de fluxo entre dois equipamentos.

3. Camada de Rede - provê comunicação fim-a-fim entre equipamentos dos usuários, usando uma ou mais redes de comunicação.

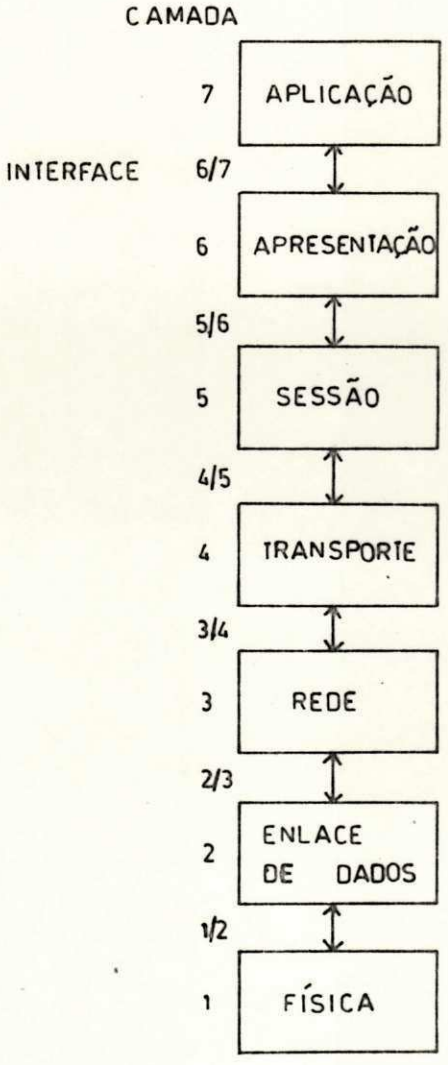


FIGURA 1.1 ARQUITETURA RM/OSI

4. Camada de Transporte - provê um melhoramento nos serviços fim-a-fim oferecendo sincronização, multiplexação, concatenacão e recuperação de falhas na camada de rede.

5. Camada de Sessão - provê administração da ligação lógica entre processos do usuário.

6. Camada de Apresentação - provê transformações sintáticas nos dados.

7. Camada de Aplicação - provê serviços de comunicação ao usuário. Esta é a única camada da qual o usuário se apercebe.

Segundo esta arquitetura as camadas 1, 2 e 3, num contexto de Redes a Longa Distância, especificam os procedimentos para interface entre um computador hospedeiro e o nó na sub-rede de comunicação. Estes procedimentos estão descritos na recomendação X.25 do CCITT (Comité Consultatif International Télégraphique et Téléphonique). Para termos acesso a sub-rede três alternativas podem ser vislumbradas:

a) Implementar a totalidade do X.25 no hospedeiro.

b) Implementar o X.25 em um processador "front-end" que agiria em benefício do hospedeiro na sua comunicação com a rede (i.e., o "front-end" utilizaria o X.25 em suas interações com a rede e o protocolo de comunicação adequado com o

hospedeiro).

c) Utilizar um conversor de protocolo tipo PAD - o qual permite também a conexão de terminais modo caractere. O PAD do lado do hospedeiro (e/ou terminais) se vale do protocolo X.28. Esta foi a alternativa utilizada neste trabalho.

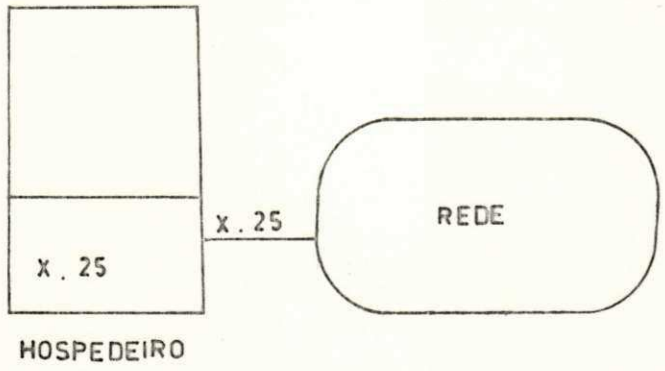
As três alternativas de conexão acima são ilustradas na figura 1.2.

As camadas 1, 2, 3 e 4 já têm padrões internacionais. Em 1983 a ISO concluiu sua proposta para um padrão internacional para o protocolo de sessão. As demais camadas ainda estão em discussão nos orgãos de padronização.

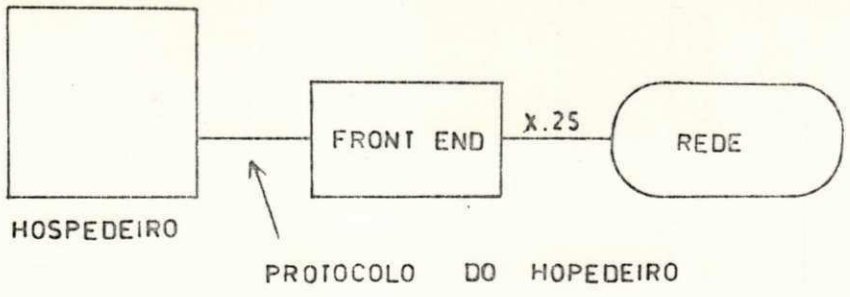
No Brasil, a SEI adotou o RM/OSI da ISO sendo seguida por instituições como a EMBRATEL e ABICOMP. A EMBRATEL é responsável pela prestação dos serviços RENPAC (Rede Nacional de Comutação de Pacotes), tendo também participado com a SEI no projeto CEPINNE.

O projeto CEPINNE tinha a intenção de interligar os computadores das Universidades Federais do Norte e Nordeste. Pretendia-se com isso adquirir experiência no projeto de redes a longa distância e depois da implantação da mesma, no desenvolvimento de aplicações como correio eletrônico, transerência de arquivos, etc [CEPI PTR].

a)



b)



c)

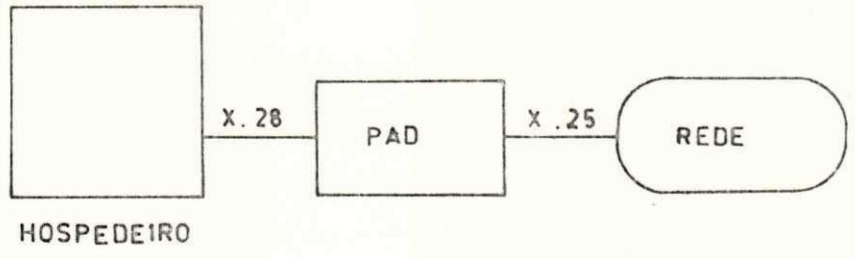


FIG 1.2 ALTERNATIVAS DE CONEXÃO

O protocolo utilizado pela rede seria o padrão X.25 da ISO/CCITT. Um conversor X.25 seria entregue a cada Universidade e estas deveriam desenvolver a interface entre o conversor e o computador utilizado e os protocolos das camadas superiores. Inicialmente, apenas as Universidades Federais de Pernambuco (Recife) e da Paraíba (Campina Grande) seriam interligadas. No Recife teríamos um computador DEC-10 rodando TOPS-10 e em Campina Grande um PDP-11/34 com UNIX versão 6.7, que seriam ligados ao nó de comutação localizado no Recife.

Aqui apresentamos e discutimos a implementação da interface (IX25) entre o Conversor X.25 (CX25) e a Estação de Transporte (ET) desenvolvida no PDP-11/34 para a rede CEPINNE.

Em termos de conteúdo, a dissertação é organizada da forma seguinte. O capítulo 2 descreve detalhadamente os sinais trocados entre o CX25 e a IX25, e entre a ET e a IX25. No capítulo 3 faremos uma especificação para fins de implementação da IX25 a partir de um modelo de Máquina de Estados Finita, modelo base para a nossa implementação. A filosofia de implementação bem como a implementação propriamente dita serão mostradas no capítulo 4. No capítulo 5 apresentaremos como foram feitos os testes na IX25. O capítulo 6 ocupa-se de uma análise crítica do trabalho, mostrando as dificuldades encontradas e apresentando sugestões para a sua extensão.

CAPÍTULO 2: SINAIS TRATADOS PELA INTERFACE X.25 (IX25)

Descreveremos em detalhes, neste capítulo, os sinais trocados pela Interface X.25 (IX25) com a Estação de Transporte (ET) e com o Conversor X.25 (CX25), (vide figura 2.1). Na primeira parte falaremos das primitivas passadas pela ET, e na segunda parte abordaremos as mensagens e os comandos do CX25. No próximo capítulo veremos como estes sinais estão correlacionados, através de um Modelo de Máquinas de Estados Finita da IX25.

2.1. Primitivas da Estação de Transporte

Os sinais descritos nesta seção estão baseados na especificação do Protocolo de Transporte da PUC [PUC 82], eles são utilizados pela Estação de Transporte para solicitar ao nível de pacotes o estabelecimento, limpeza e encerramento de circuitos virtuais e também para ser informada de algum problema com qualquer circuito virtual para poder ativar os mecanismos de recuperação que achar necessários. A comunicação da ET com a IX25 é feita através de primitivas que fornecem os parâmetros que vão ser trocados. Esses parâmetros são divididos em dois tipos:

de entrada	passados da ET para a IX25
de saída	no sentido contrário

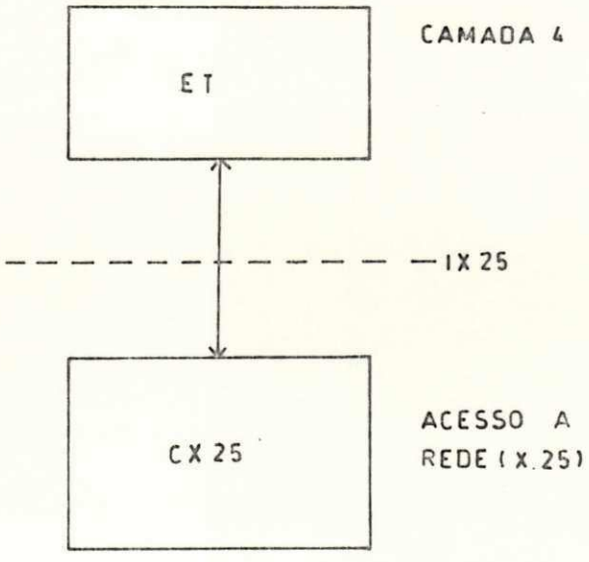


FIG 2.1 - LOCALIZAÇÃO DA INTERFACE

As primitivas podem ser divididas em dois grupos:

a) Para gerência de Circuito Virtual (CV)

ESTABCV	pedido de estabelecimento de CV
LIMPACV	pedido de desconexão do CV
RESETCV	pedido de reinicialização do CV

b) Para transmissão e recebimento de informação

TRAPACOTE	transmissão de pacotes
RCPACOTE	recebimento de pacotes
RCSINAL	recebimento de sinais do nível 3

Passamos agora a uma descrição mais detalhada de cada primitiva.

2.1.1. ESTABCV

Solicita ao nível de pacotes o estabelecimento de um CV. A informação de que a operação foi bem sucedida é uma informação local não significando que o CV foi conseguido.

Parâmetros de entrada (->):

-> Endereço do ETD remoto

Parâmetros de saída (<-):

<- Número do canal lógico.

<- Código de Retorno:

00 operação bem sucedida

01 endereço DTE remoto inválido ou não acessível

02 não há canal lógico disponível localmente

2.1.2. LIMPACV

Solicita ao nível de pacotes o encerramento do CV.

Parâmetros de entrada:

-> número do canal lógico

Parâmetros de saída:

<- Código de Retorno

00 operação bem sucedida

10 número do canal lógico inválido ou não associado com CV

11 processando comando do nível 3

12 CV sendo estabelecido

2.1.3. RESETCV

Solicita reinicialização do CV, o que implica que os pacotes que estiverem em transmissão ou aguardando transmissão serão descartados.

Parâmetros de entrada:

-> número do canal lógico

Parâmetros de saída:

<- Código de retorno

00 operação bem sucedida

20 número do canal lógico inválido ou não associado com CV

21 CV sendo encerrado

22 CV sendo reinicializado

23 processando comando nível 3

24 CV sendo estabelecido

2.1.4. TRAPACOTE

Entrega ao nível de pacotes os dados a serem transmitidos pelo CV.

Parâmetros de entrada:

-> número do canal lógico

-> endereço do "buffer" onde estão os dados a serem transmitidos.

-> tamanho do "buffer"; informa o número de "bytes" a serem transmitidos.

Parâmetros de saída:

<- Código de retorno

00 operação bem sucedida

40 número do canal lógico inválido ou não associado com CV

41 endereço do "buffer" inválido ou não associado com CV

42 tamanho do "buffer" excede o máximo permitido

43 CV sendo estabelecido

44 CV sendo reinicializado

45 CV sendo encerrado

46 nível de pacote ocupado

47 esperando "prompt" de dados

48 erro no dispositivo de E/S

2.1.5. RCPACOTE

Solicita ao nivel de pacotes os dados recebidos e já disponiveis para entrega.

Parâmetros de entrada:

-> endereço do "buffer" onde os dados devem ser colocados.

Parâmetros de saida:

<- número de "bytes" transferidos

<- número do canal lógico

<- Código de retorno

00 operação bem sucedida

50 endereço do "buffer" invâlido ou não acessivel

51 não há dados para entregar

52 CV não está disponivel para transferência de dados

2.1.6. RCSINAL

Solicita ao nível de pacotes a entrega de qualquer informação relevante e que exija alguma ação por parte do nível de transporte. Esta primitiva também pode servir para recebimento de interrupção [PUC 82].

Parâmetros de entrada:

-> não há

Parâmetros de saída:

<- número do canal lógico ao qual se refere a sinalização

<- código de causa e diagnóstico para o evento sinalizado

<- o endereço do ETD remoto

<- "byte" de interrupção *

<- código de sinalização

00 nada a informar

01 interrupção *

* - Para o caso de haver interrupção

02 CV encerrado

04 CV reinicializado

08 CV estabelecido

2.2. Mensagens e Comandos do Nível 3

A descrição dos sinais a nível de pacotes está baseada nos manuais de utilização do Conversor X.25 [CX25 MO] [CX25 MU] [CX25 ED]. A Interface X.25 se comunica com o Conversor X.25 por meio de mensagens (transmitidas no sentido conversor-interface), de comandos (sentido contrário) e de troca de dados (em ambos os sentidos). Utiliza-se uma versão simplificada da Recomendação X.28, vide figura 2.2.

Os comandos utilizados são os seguintes:

CR enviar comandos na fase de conexão

ESC enviar comandos na fase de troca de dados

SEL comando de seleção para pedido de conexão

CLR comando de pedido de desconexão

RESET comando de pedido de reinicialização

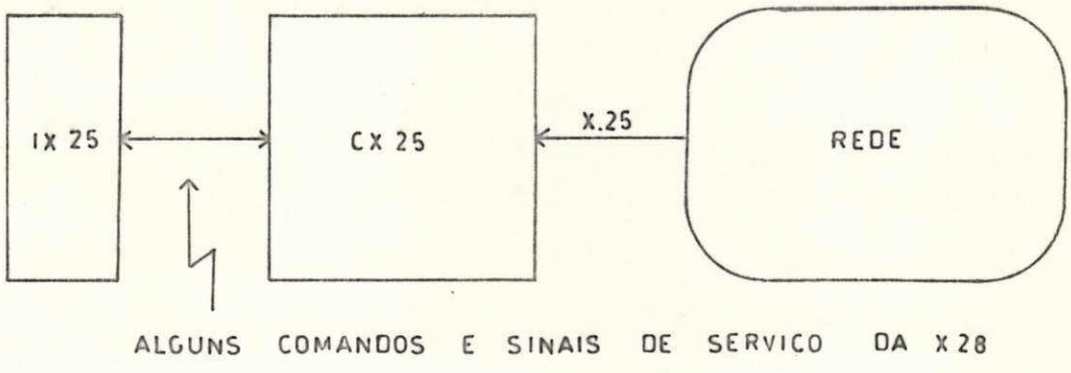


FIG 2.2 - PROTOCOLO UTILIZADO NO CX.25

As mensagens de serviço utilizadas são:

FORMAT EFFECTOR	formatador de linha ou confirmação de RESET
COM	indicação de conexão estabelecida
ATENCAO	indicação de pedido de conexão do lado remoto
CLR	indicação de desconexão
CLR CONF	indicação de confirmação de desconexão
RESET	indicação de reinicialização
ERR	indicação de erro de comando
>	indicação de "prompt" para comandos
*	indicação de "prompt" para dados
OCUPADO	indicação de nível 3 ocupado
DESOCUPADO	indicação de nível 3 desocupado

Troca de dados:

DADOS dados do usuário

Neste ponto, cabem algumas explicações sobre o funcionamento do CX25.

Temos dois modos distintos de operação do CX25:

a) modo de Comandos

b) modo de Transferência de Dados

Estas dois modos podem ser distinguidas um do outro através de "prompts" diferentes, "*" para Transferência de Dados e ">" para Comandos. Ao ser ligado ou após um RESET externo, o CX25 espera que o ETD local transmita um "CR" (CARRIAGE RETURN) para passar para fase de comandos.

A cada ETD local corresponde um canal lógico da seguinte forma:

ETD IV - canal lógico 1
ETD III - canal lógico 2
ETD II - canal lógico 3
ETD I - canal lógico 4

Depois que um CV é estabelecido, entra-se na fase de transferência de dados; se quisermos enviar qualquer comando durante essa fase temos que enviar o caractere "ESCAPE" para

o CX25, que pode ser enviado no início ou no meio de uma linha de dados. Neste último caso, a linha de dados é enviada como está.

Para a IX25 distinguir os dados recebidos dos sinais de serviço, o caractere "\" é enviado antes dos caracteres do sinal de serviço. Assim quando estivermos na fase de transferência de dados, mensagens e informações terão a seguinte forma:

<MINF> ::= <<dados>><FORMAT EFFECTOR> |
<\><<sinal de serviço>>

Todo sinal de serviço é terminado com um "format effector" permitindo sua delimitação. Abordaremos isso em detalhes, mais adiante.

Examinemos melhor os comandos e as mensagens acima. Utilizaremos a "Forma Normalizada de Backus (Descrição Sintática Recursiva, [DONO 72])" para descrição dos mesmos. Foi realizado aqui um melhoramento na documentação do CX25 no sentido de um maior rigor na descrição e introduzimos na seção de comandos o envio dos caracteres de "Retorno do Carro" e de "Escape", na seção de mensagens o "Format Effector" como confirmação de RESET e uma nova seção descrevendo a troca de dados.

Símbolos não terminais são escritos entre '<>'. O sinal ::= é lido como "é substituído por". Alternativas são separadas por uma barra vertical "|" (leia-se "ou").

2.2.1. Comandos

2.2.1.1. Retorno do Carro (CR)

Este comando é utilizado para se entrar na fase de comandos quando do início de um pedido de conexão.

<CR>::=caractere ASCII de retorno do carro.

2.2.1.2. Escape (ESC)

Este comando é também utilizado para se entrar na fase de comandos, mas quando estamos na fase de transferência de dados.

<ESC>::=caractere ASCII de escape.

2.2.1.3. Pedido de Conexão (PCON)

Este comando é utilizado para pedir o estabelecimento de um CV, e só será executado se este já não estiver estabelecido.

<PCON>::=SEL:<BE>*<DU><CR> | SEL:<BE><CR>

<BE>::=endereço do ETD remoto (3 ou 6 dígitos decimais em ASCII - 3 dígitos de endereço e 3 dígitos de campo complementar, se existir).

<DU>::=até 16 caracteres de dados de identificação do ETD local.

2.2.1.4. Pedido de Desconexão (PDESCON)

Este comando desfaz a ligação entre o ETD local e o ETD remoto.

<PDESCON>::=CLR<CR>

2.2.1.5. Pedido de Reinicialização (PRES)

Este comando reinicializa o CV.

<PRES>::=RESET<CR>

OBSERVAÇÃO

Durante o estabelecimento e reinicialização da conexão, nenhum novo comando/dados será aceito até que chegue a resposta ao comando anterior.

2.2.2. Mensagens

Todas as mensagens, exceto os sinais de aviso de "prompt", são precedidos pelo caractere "\" para diferenciar as mensagens dos dados do usuário.

As mensagens são as seguintes:

2.2.2.1. Mensagens de Erro (MERR)

Indica que houve erro no comando enviado.

<MERR>::=ERR<SP><N><FE>

<SP>::=caractere ASCII espaço.

<N>::=1|2|3

<FE>::=<CR><LF><sequência de caracteres nulo>

<LF>::=caractere de mudança de linha.

Os erros (<N>) indicam o seguinte:

1 erro de sintaxe do comando.

2 pedido de conexão já estabelecida.

3 pedido de desconexão ou reinicialização numa conexão ainda não estabelecida.

2.2.2.2. Sinal de Serviço de Conexão Pronta (SSCP)

Confirmação do pedido de conexão.

<SSCP>::=COM<FE>

2.2.2.3. Sinal de Serviço de Conexão Remota (SSCR)

Indica a chegada de um pedido de conexão vindo de um ETD remoto.

<SSCR>::=ATENCAO:<SP><BER><SP>COM<DUR><FE>

<BER>::=endereço do ETD chamador (3 dígitos decimais em ASCII).

<DUR>::=dados do ETD chamador (de 0 a 16 caracteres).

2.2.2.4. Sinal de Serviço de Indicação de Desconexão (SSID)

Indica que um pedido de desconexão foi enviado do ETD remoto ou da Rede.

<SSID>::=CLR<SP><causa><SP><diagnóstico><FE>

<causa>::=OCC|NC|INV|VA|ERR|RPE|NP|DER|DTE

<diagnóstico>::=de 1 a 3 dígitos em ASCII.

A causa e o diagnóstico seguem a codificação X.25.

Descrição dos mnemônicos de causa [RENP 83]:

OCC O ETD chamador não está apto para receber chamadas para estabelecimento de conexão.

NC Existência de certas condições na rede como:

1) Congestionamento temporário da Rede.

2) Uma falha temporária na Rede.

INV Pedido de facilidade inválido.

NA O ETD chamador não tem permissão para se conectar ao ETD chamado (grupo fechado de usuários).

ERR Erro de procedimento causado pelo ETD e detetado pelo PAD.

RPE Erro de procedimento causado pelo ETD e detetado pela interface ETD/ECD remota.

NP Endereço não designado para nenhum ETD.

DER O número chamado está fora de ordem.

DTE O ETD remoto fechou a chamada.

2.2.2.5. Sinal de Serviço de Confirmação de Desconexão (SSCD)

Confirma o pedido de desconexão.

<SSCD>::=CLR<SP>CONF<FE>

2.2.2.6. Sinal de Serviço de Reinicialização (SSR)

Indica que um pedido de reinicialização foi enviado do ETD remoto ou da Rede.

<SSR>::=RESET<SP><rcausa><SP><diagnóstico><FE>

<rcausa>::=DTE|ERR|NC

2.2.2.7. Sinal de Serviço de Aviso de "Prompt" (SSAP)

Indica quando podemos enviar dados ou quando podemos enviar comandos.

<SSAP>::=<prompt><FE>

<prompt>::=<comando>|<dados>

<comando>::=>

<dados>::=*

2.2.2.8. Sinal de Serviço de Nível 3 Ocupado (SSNO)

Indica que o CX25 não pode mais receber dados do ETD local.

<SSNO>::=NIVEL<SP>3<SP>OCUPADO<FE>

2.2.2.9. Sinal de Serviço de Nível 3 Desocupado (SSND)

Indica que o CX25 foi liberado para receber dados.

<SSND>::=NIVEL<SP>3<SP>DESOCUPADO<FE>

2.2.2.10. Sinal de Serviço de Confirmação de Reinicialização (CRES)

Confirma um pedido de reinicialização.

<CRES>::=<FE>

2.2.3. Troca de Dados

2.2.3.1. Dados enviados para o ETD Remoto (DETDR)

São dados do usuário para serem enviados para o ETD remoto.

<DETDR>::=<linha>|<linhaCR>

<linha>::=0 a 128 bytes de dados.

<linnaCR>::=0 a 127 bytes de dados seguidos por <CR>.

2.2.3.2. Dados Recebidos do ETD Remoto (DREC)

São os dados que foram enviados pelo CX25 remoto.

<DREC>::=<linha><FE>|<linhaCR><LF><sequência de nulos>

Observação: Se o <CR> (fim de bloco) foi enviado nos dados não será enviado novamente no <FE>.

CAPÍTULO 3: ESPECIFICAÇÃO PARA FINS DE IMPLEMENTAÇÃO DA IX25

Mostraremos agora como foi realizada a especificação da IX25 com a finalidade de implementação, nesse sentido a nossa especificação carece de um formalismo rigoroso, pois parte da implementação já estava definida intuitivamente e ainda a interação especificação-implementação provocou varias mudanças tanto numa como na outra.

O nosso trabalho foi dificultado pela falta de clareza da documentação do CX25 [CX25 MO] [CX25 MU]. Os seguintes pontos duvidosos foram levantados:

1) O comando RESET como foi especificado realiza a mesma coisa que um pedido de desconexão. Optamos por uma reinicilização dos "buffers" sem perda do Circuito Virtual.

2) Não existe confirmação para o comando RESET. Utilizamos a mesma da Recomendação X.28, um "FORMAT EFFECTOR".

3) Existia o comando de interrupção, mas não tinha indicação de que uma interrupção chegara.

4) Não ficou claro como se faria a inicialização de uma conexão, se um caratere de 'CR' ou de 'ESC' deveria ser enviado para se entrar na fase de comandos. Escolhemos o primeiro.

5) Os dados vindos da ET não podiam ser enviados se não fossem mascarados. Assunto discutido em detalhes no capitulo 4.

Algumas destas dificuldades só seriam sanadas quando da chegada do CX25, até lá, nós fariamos suposições de como seria o funcionamento do mesmo. Este fato nos levou a nossa Máquina de Estados Finita (MEF) aqui especificada, que acreditamos ser uma maneira mais clara de se apresentar uma documentação, principalmente para fins de implementação.

Uma MEF é definida como sendo uma quintupla [DANT 80]:

$$< E, I, S, FPE, FS >$$

onde,

E é o conjunto finito de estados;

I é o conjunto finito de entradas;

S é o conjunto finito de saidas;

FPE é uma função de transição de estados;

$$FPE : I \times E \rightarrow E ;$$

FS é uma função de saida

$$FS : I \times E \rightarrow S ;$$

FPE e FS descrevem o comportamento da máquina. Se num dado estado, uma entrada é recebida a função de saída indicará a saída gerada e a função de transição de estado indicará o próximo estado da máquina.

O conjunto de entradas é formado pelas primitivas da ET e pelas mensagens vindas do CX25. O conjunto de saídas é constituído pelos parâmetros de saída das primitivas da ET e pelos comandos enviados ao CX25. Tanto o conjunto de entradas como o de saídas estão definidos no capítulo anterior.

Definimos os seguintes estados em que a IX25 pode estar em relação a um CV.

DISPONIVEL CV disponível.

ESPPPTSEL esperando "prompt" de comando para SEL.

SNDESTAB sendo estabelecido.

ESPPPTDAD esperando "prompt" de dados.

DADOS troca de informação entre usuários.

N3OCP nível 3 ocupado.

ESPPPTRST esperando "prompt" de comando para RESET.

SNDRSTCMD sendo "resetado" por comando.

ESPPPTCLR esperando "prompt" de comando para CLR.

SNDENCRR sendo encerrado.

Outros estados poderiam ser definidos levando-se em conta a situação dos "buffers" internos da IX25 mas isto levaria a uma quantidade maior de estados, o que dificultaria uma visão geral do encadeamento de transições, prejudicando a implementação.

Passaremos agora para a descrição de cada um dos estados e de suas possíveis transições. Mostraremos como a partir do estado descrito poderemos alcançar os estados adjacentes ao mesmo.

No diagrama de estados adotaremos a nomenclatura mostrada na figura 3.1.

Estado DISPONIVEL - Significa que o CV não está sendo utilizado, portanto disponível para uma conexão, vide figura 3.2.

Estado ESPPPTSEL - é o estado em que se espera um "prompt" de comandos para enviar um pedido de conexão, vide figura 3.3.

Estado SNDESTAB - é o estado em que um pedido de conexão foi enviado e se espera a confirmação do mesmo, vide figura 3.4.

Estado ESPPPTDAD - é o estado em que se recebeu uma

indicação de que uma conexão foi estabelecida, ou em que se enviou um pacote de dados, e um "prompt" de dados é esperado para se poder enviar dados, vide figura 3.5. A variável tambuftrans que aparece na figura indica se o "buffer" utilizado para transmissão de pacotes está cheio (diferente de zero) ou vazio (igual a zero).

Estado DADOS - significa que os dados dos usuários que estão conectados podem ser transferidos, vide figura 3.6.

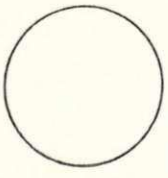
Estado N3OCP - é o estado em que o CX25 avisa que não pode mais receber dados da IX25, vide figura 3.7.

Estado ESPPPTRST - é o estado no qual se deseja enviar um comando de reinicialização e um "prompt" de comandos é esperado, vide figura 3.8.

Estado SNDRSTCMD - é o estado no qual um pedido de RESET foi enviado ao CX25 e se espera a confirmação do mesmo, vide figura 3.9.

Estado ESPPPTCLR - é o estado em que se quer enviar um comando de desconexão e um "prompt" de comando é esperado, vide figura 3.10.

Estado SNDENCRR - é o estado em que um pedido de desconexão foi enviado e se espera a confirmação do mesmo, vide figura 3.11.



REPRESENTA UM ESTADO DA MÁQUINA

INDICA O SENTIDO DA TRANSIÇÃO

RECEPÇÃO DE UMA PRIMITIVA OU DE UMA MENSAGEM

AÇÃO EXECUTADA (ENVIO DE UM COMANDO)

FIG 3.1 - NOMENCLATURA DA MEF

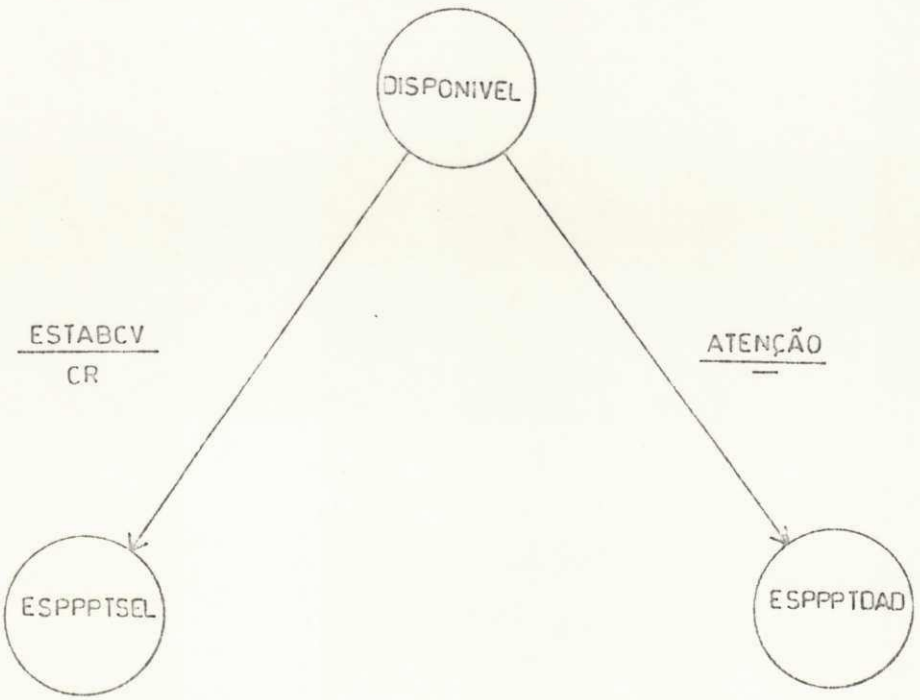


FIG 3.2 - ESTADO DISPONIVEL

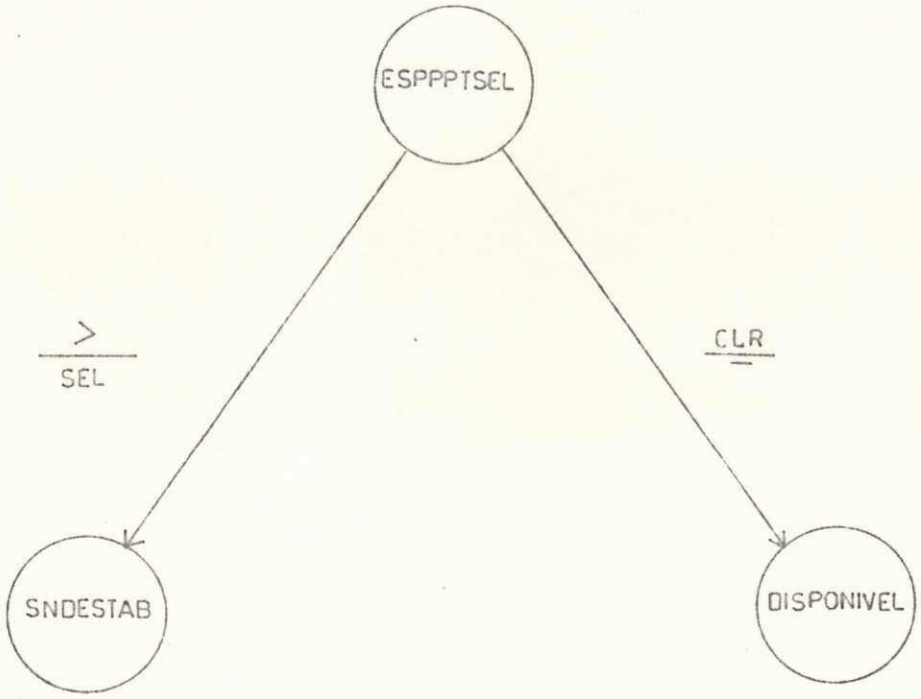


FIG 3.3 - ESTADO ESPPPTSEL

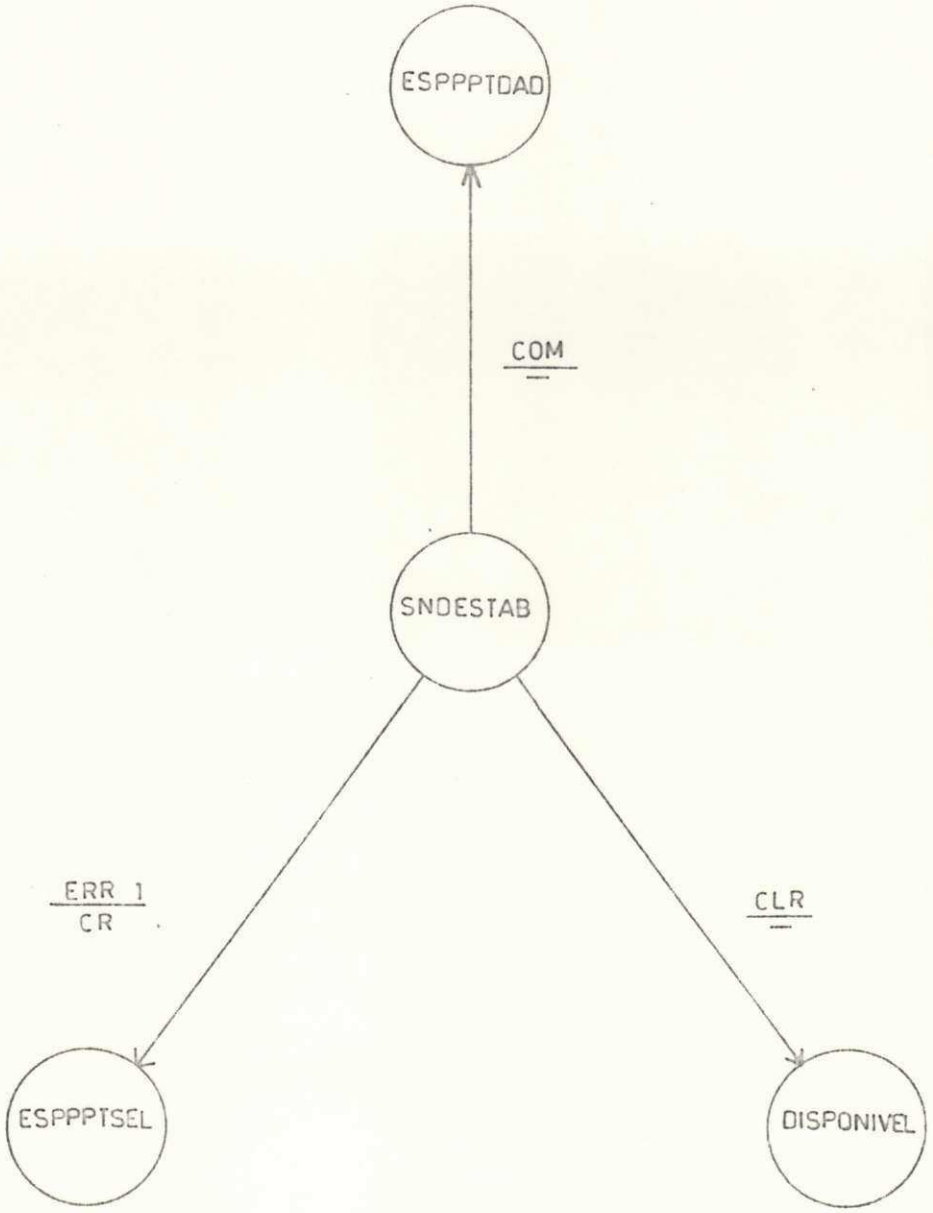


FIG 3.4 - ESTADO SNDESTAB

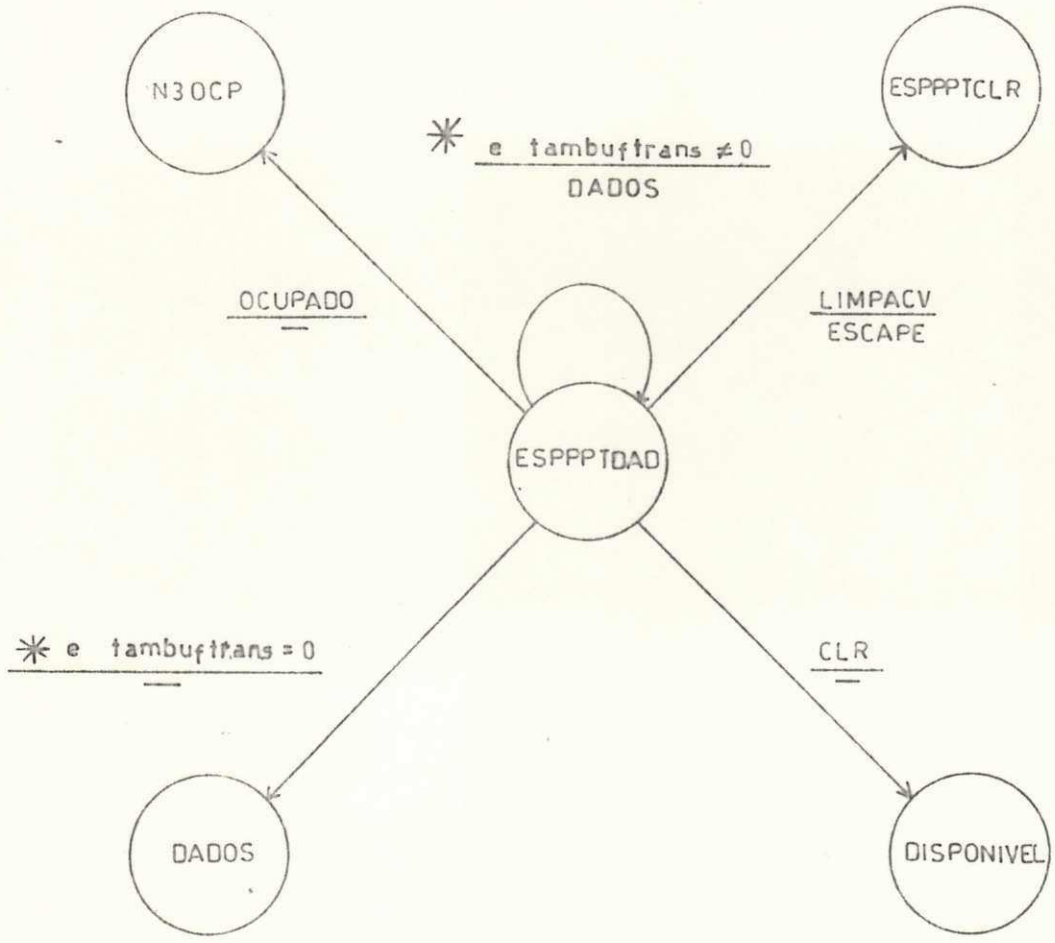


FIG 3.5-ESTADO ESPPPTDAD

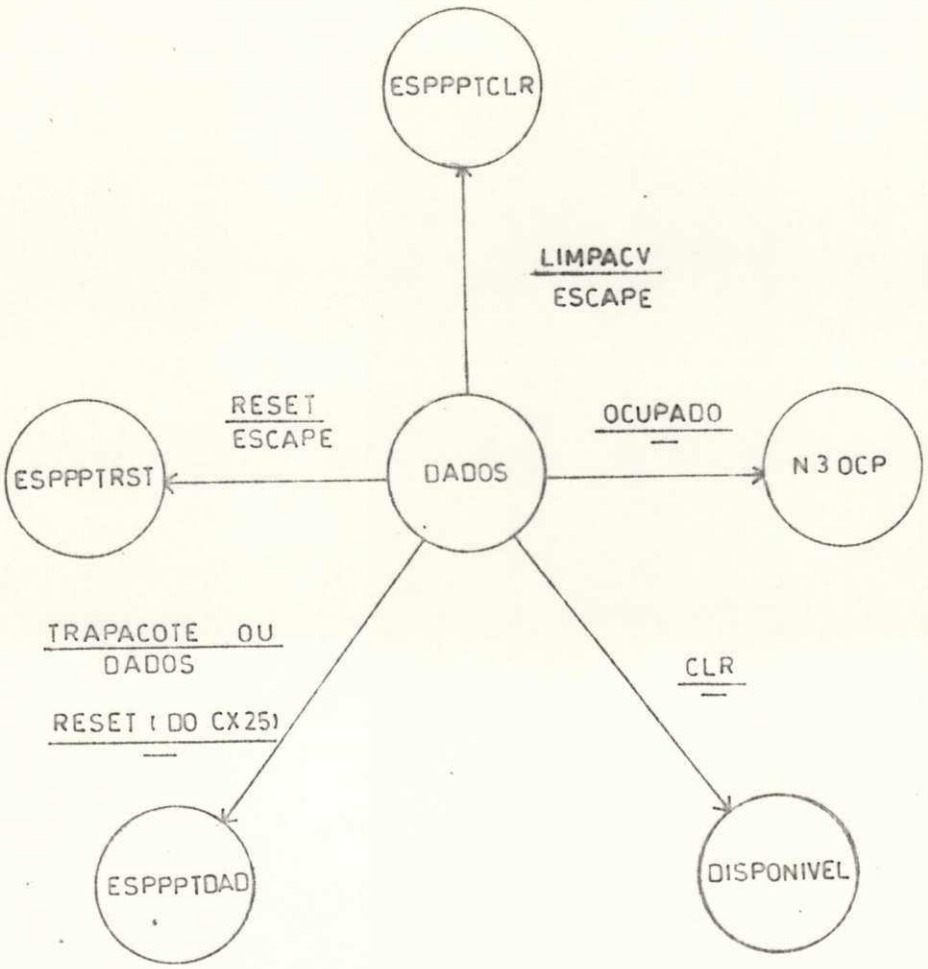


FIG 3.6 - ESTADO DADOS

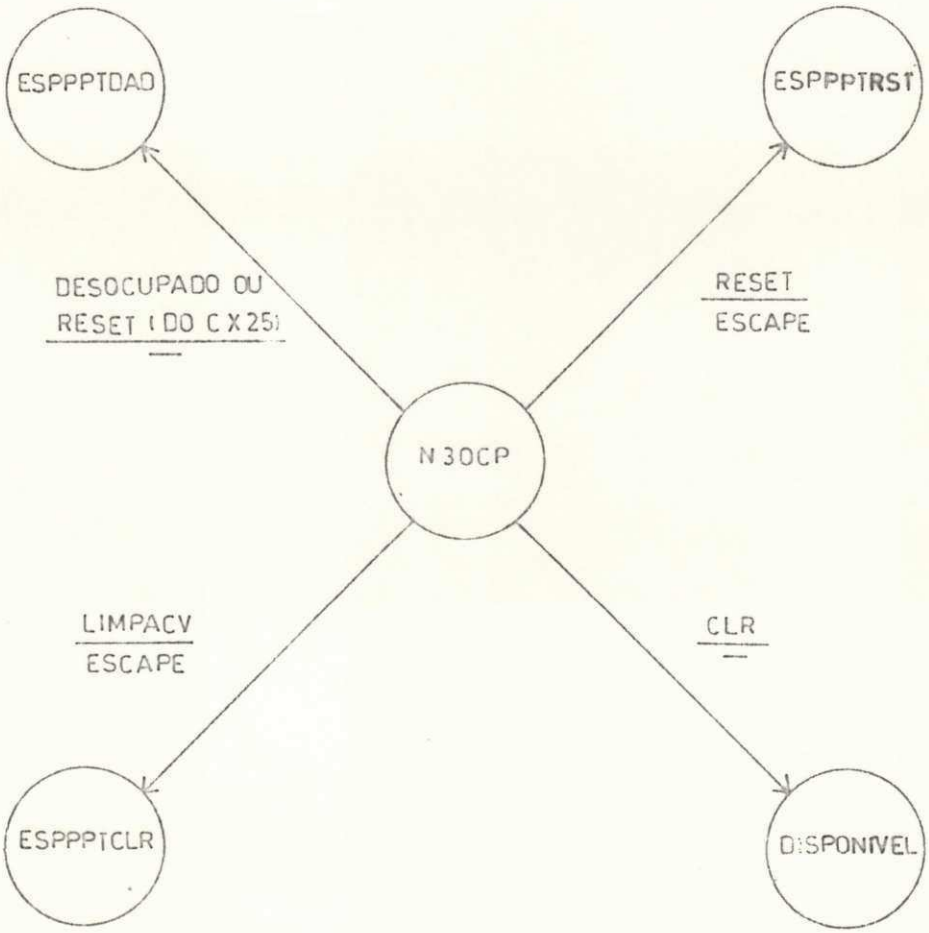


FIG 3.7 - ESTADO N30CP

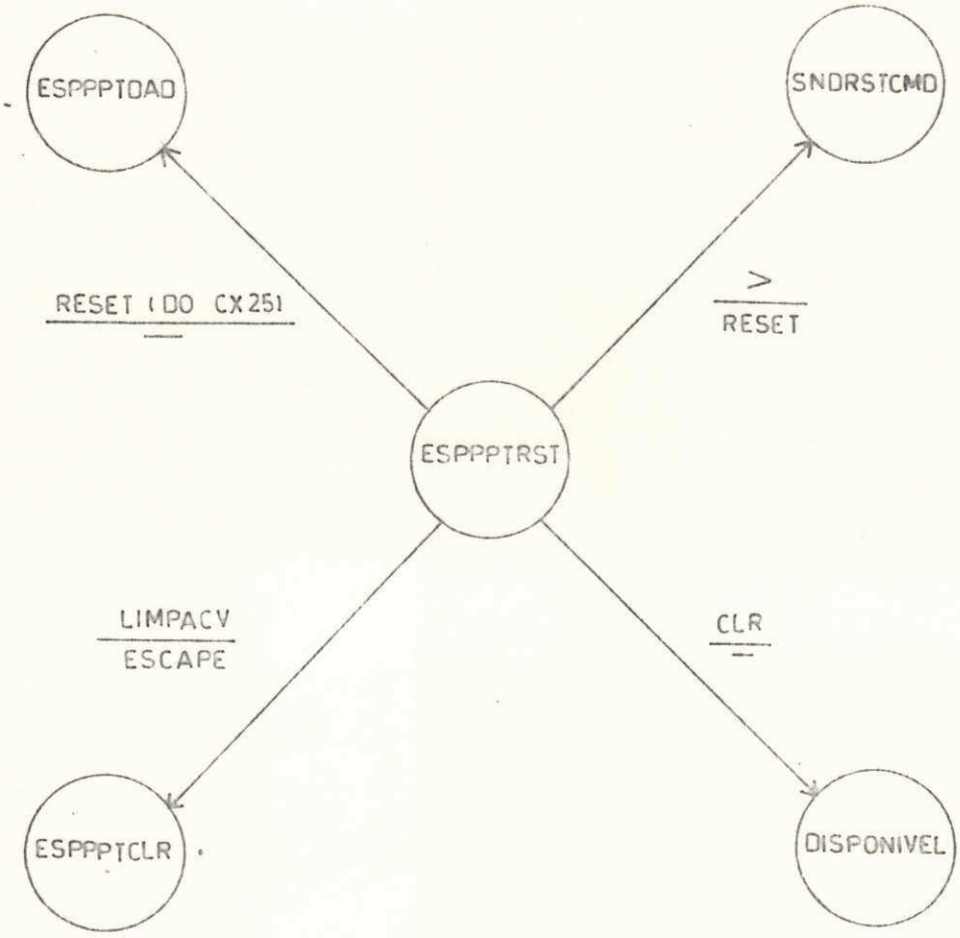


FIG 3.8 - ESTADO ESPPPTRST

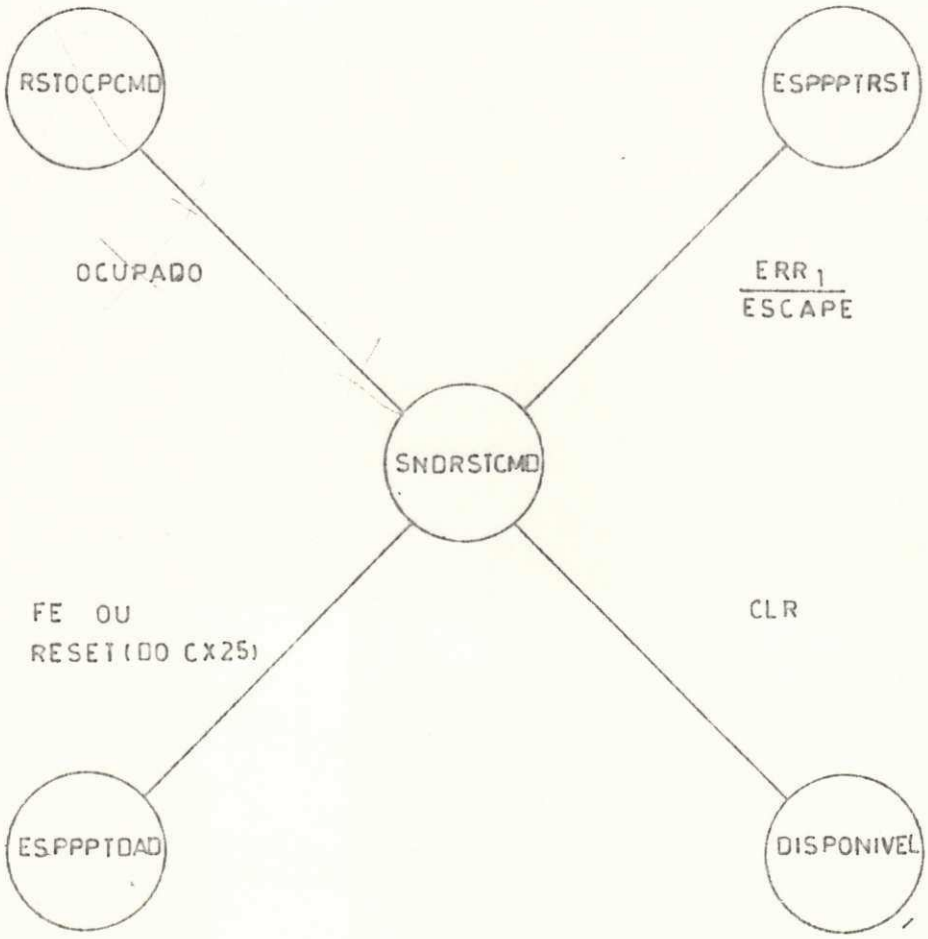


FIG 39 - ESTADO SNDRSTCMD

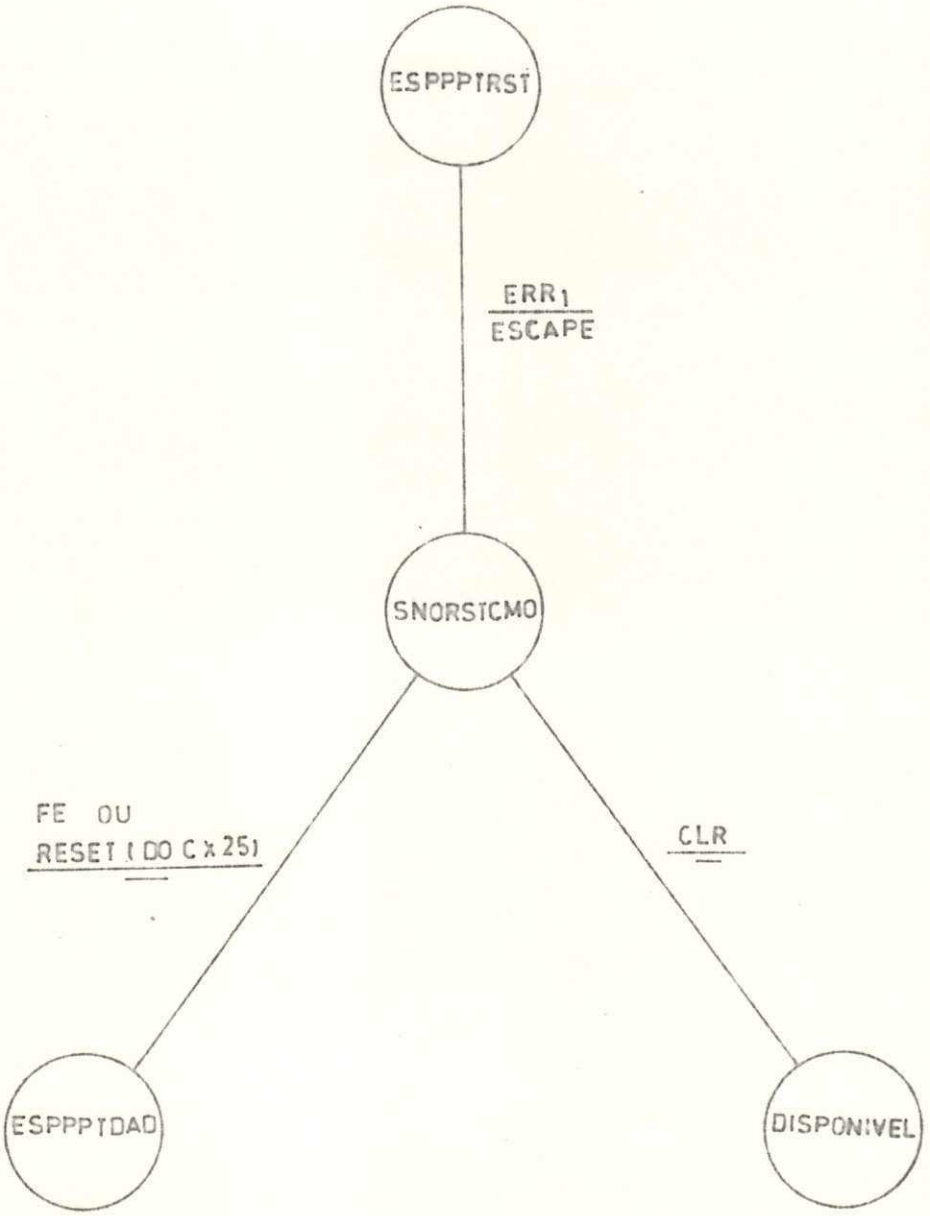


FIG 39 - ESTADO SNORSTCMD

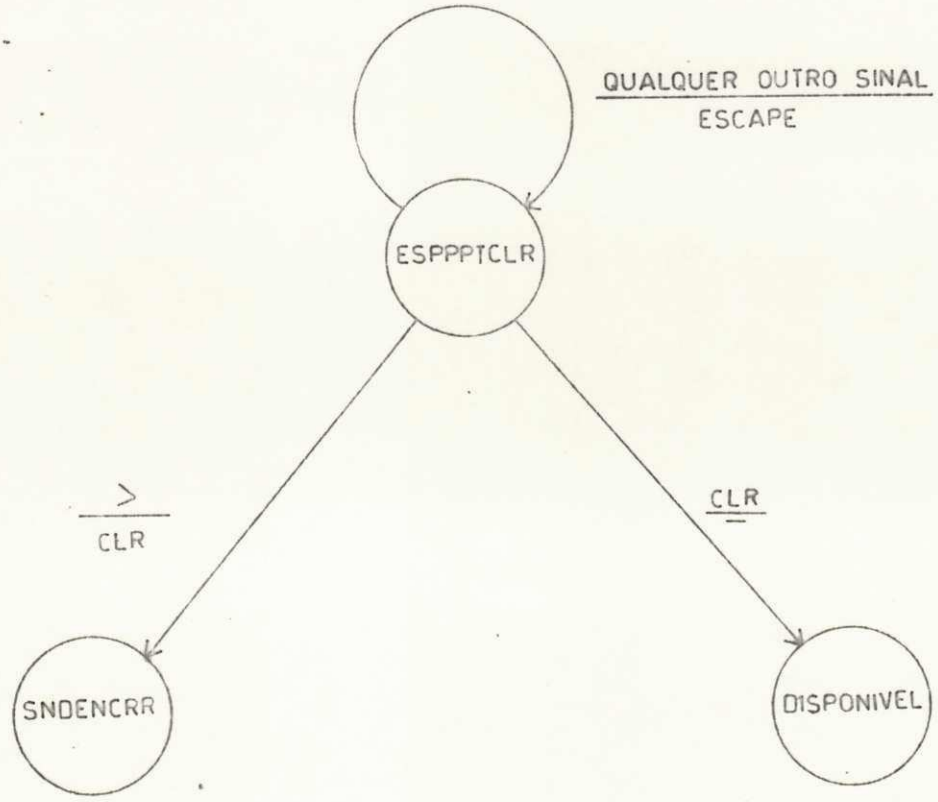


FIG 3.10 - ESADO ESPPPTCLR

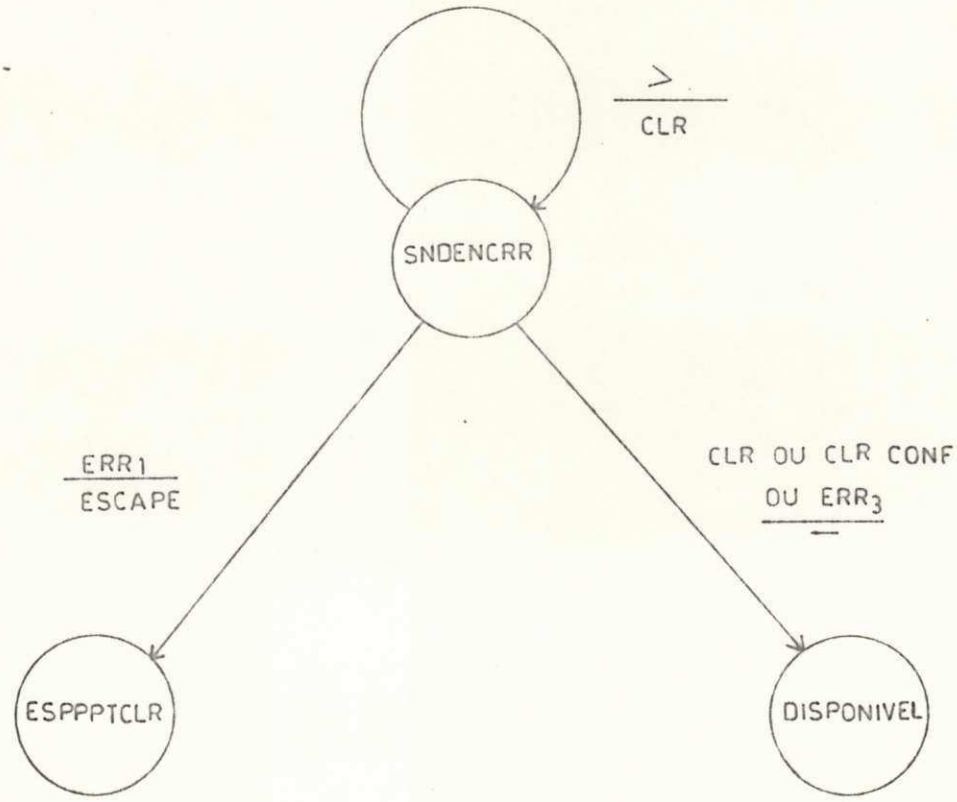


FIG 3.11 - ESTADO SNDENCRR

Como já dissemos no inicio do capitulo, rigorosamente a IX25 não está totalmente especificada com o nosso modelo, alguns aspectos ainda precisam ser esclarecidos. Por exemplo, as primitivas RCSINAL e RCPACOTE não aparecem no diagrama de estados, elas não provocam mudanças nos estados que definimos mas mexem nos "buffers" internos da IX25; RCSINAL pode ser disparada em qualquer estado e limpa o "buffer" de sinalização; RCPACOTE é disparada nos estados ESPPPTDAD, N3OCP ou DADOS e limpa o "buffer" de recepção de dados; o envio e o recebimento de dados tem uma forma peculiar. Estes pontos serão discutidos no próximo capítulo no qual descrevemos como foi realizada a implementação da IX25.

CAPÍTULO 4: IMPLEMENTAÇÃO DA IX25

Este capitulo se ocupa em discutir como foi implementada a IX25 e a filosofia de implementação empregada. Utilizamos a linguagem de programamção "C". Os nomes das variáveis de maior interesse utilizadas pelos programas, bem como a localização dos mesmos no sistema de arquivos do UNIX serão fornecidos, com a intenção de facilitar uma possivel continuação do trabalho. Os arquivos mencionados têm o seguinte percurso comum /u/cepinne/src/x25/ix25 na árvore do sistema de arquivos. A base de toda implementação está no modelo especificado no capitulo anterior. As listagens dos programas desenvolvidos estão no apêndice.

4.1. Ligação Física

O CX25 envia mensagens e recebe comandos para/do ETD local, no nosso caso um PDP-11/34 rodando UNIX versão 6, através de uma interface do tipo RS232C/V.24 com conectores fêmeas de 25 pinos padrão ISO DP25, no modo de transmissão assincrono.

Este tipo de interface é compativel com a interface de linha serial assincrona do PDP-11, permitindo assim uma ligação física direta com o CX25 sem a necessidade de "hardware" adicional.

4.2. Software

O "software" da IX25 pode ser visto de uma maneira genèrica, como se constituindo em três blocos principais (vide figura 4.1). Um bloco de controle que cuida de todo o controle da IX25, interpreta todas as mensagens que chegam do CX25, solicita o envio de comandos (dependendo de alguma primitiva da ET ou de alguma mensagem do CX25), cuida da tranparência dos dados trocados entre a ET e o CX25 e guarda informações para a ET. Um bloco de recebimento que se encarrega do recebimento de dados ou mensagens vindas do CX25 que são identificadas e passadas para o bloco de controle e do envio de dados ou comandos para o CX25 (dependendo do "prompt" recebido do CX25). E finalmente, um bloco de comunicação com a ET, o qual ativa o bloco de controle segundo a primitiva solicitada, a ET e a IX25 formam um único processo ativo de modo que comunicação entre as duas entidades é feita atravès da passagem de parâmetros (p.e., apontadores) entre funções.

Como já dissemos, nossa implementação tem como base um modelo de MEF. Quando um modelo deste tipo é usado para se representar um determinado sistema, a especificação do algoritmo de controle para sua implementação torna-se bastante simples.

FIG 4.1 ESTRURAÇÃO DO SOFTWARE

Em termos genéricos, uma MEF implementando um certo sistema, funciona da seguinte maneira: se o sistema estiver num determinado estado A e receber uma entrada I, produzirá uma saida S (que poderá ser nula), e irá para um estado B (que poderá ser o mesmo estado anterior).

Basicamente o nosso algoritmo terá os seguintes passos:

1. Ler a entrada

2. Gerar a saida

3. Gerar o próximo estado

4. Ficar esperando uma nova entrada

Com algumas variações, a nossa implementação estará bem próxima do algoritmo acima. O conjunto de entradas é formado pelas Primitivas que são na realidade funções chamadas pela ET, e pelas Mensagens oriundas do CX25. Vale a pena observar que as funções que representam as primitivas têm parâmetros de entrada e, rigorosamente, o valor assumido por cada parâmetro deveria ser considerado como uma entrada. Isto provocaria uma explosão do número de entradas, assim optamos por tratar os parâmetros das funções antes de chamarmos as funções de saida e de próximo estado, modificando um pouco nosso algoritmo básico.

As mensagens que chegam do CX25 são processadas para serem entregues às funções de saida e de próximo estado. O conjunto de saidas é formado pelos códigos de retorno e

parâmetros de saída das primitivas e pelos comandos enviados ao CX25, descritos no capítulo 2.

A função de saída foi implementada como sendo uma matriz bidimensional (figura 4.2) cujas linhas são as entradas e cujas colunas são estados.

Cada membro da matriz é uma estrutura como mostrado abaixo:

```
struct saida {
     int (*func)(); /* apontador para uma função */
     int pl;
};
```

De forma semelhante implementamos a função de próximo estado. Agora, entretanto, os membros da matriz representam os próximos estados (figura 4.3) a serem assumidos por nossa MEF. Assim, por exemplo, se estivermos no estado ESPPPTRST e um "prompt" de comandos (>) chega do CX25, temos que para o estado e a entrada dados a função de saída é ativada pela seguinte estrutura

{envia_cmd, RESET}

onde onde envia_cmd é um apontador para a função que envia comandos e RESET é o comando a ser enviado. A função de próximo estado indica que o próximo estado é SNDRSTCMD. Com esta estruturação conseguimos simplificar bastante o esforço de implementação devido a uma vizualização global do bloco de controle.

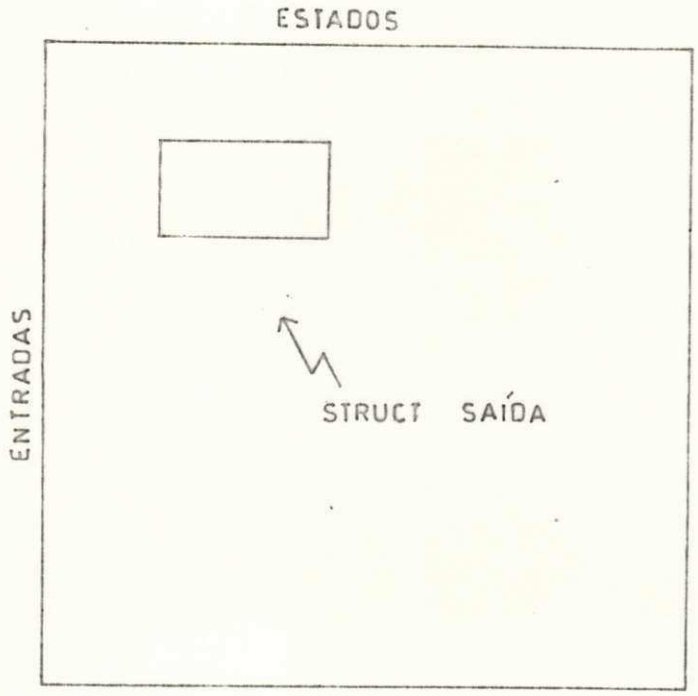


FIG 4.2 - FUNÇÃO DE SAÍDA

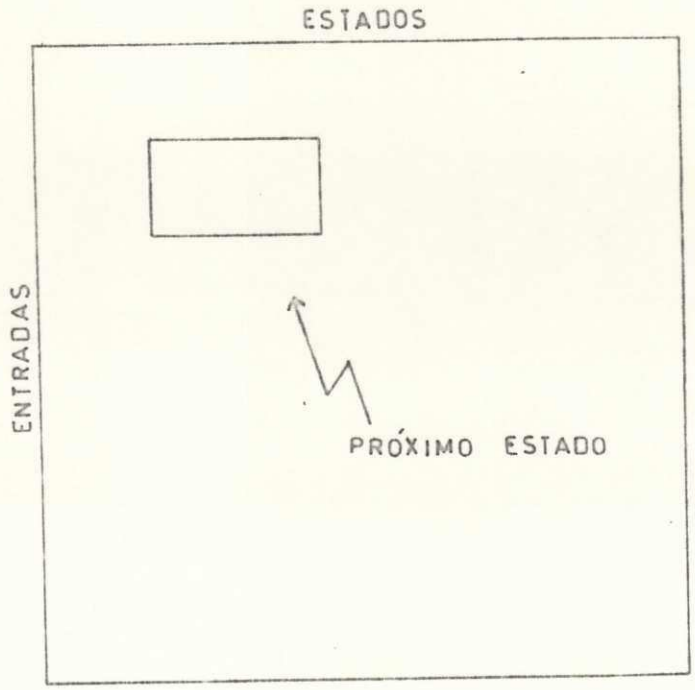


FIG 4.3 - FUNÇÃO DE PRÓXIMO ESTADO

O conjunto de entradas e o conjunto de estados são definidos no arquivo cabeçalho ix25.h e as funções de saída e de próximo estado são mostradas no arquivo cabeçalho ix25global.h.

Como o CX25 não estava disponivel os comandos serão enviados a um arquivo chamado cmd x25, o que está perto de uma implementação real, já que entrada e saida no UNIX são feitas lendo e gravando arquivos respectivamente.

4.2.1. Implementação das Primitivas

Passaremos agora a descrever a implementação das primitivas da ET, estas são funções da IX25 chamadas pela ET e que retornam informações sobre o sucesso ou não da operação solicitada, e quando for o caso alguns "buffers" são passados para esta.

4.2.1.1. Implementação de ESTABCV

Quando a função ESTABCV é chamada, ela tenta conseguir um CV para a ET, para isso ela envia um caractere de retorno do carro para o CX25 (na nossa simulação, para o arquivo cmd x25). Isto só é realizado se o CV estiver DISPONIVEL e se o endereço do ETD remoto for válido.

Para qualquer uma das situações acima, um código de retorno é devolvido ao processo chamador (ET) indicando o ocorrido.

Os passos do algoritmo da função ESTABCV são mostrados a seguir:

1. Verificar se o endereço do ETD remoto é válido e se o canal lógico está DISPONIVEL
2. Se não, executar o passo 5
3. Caso contrário, enviar um caractere de retorno do carro ao CX25
4. Mudar o estado do CV
5. Retornar um código de retorno a ET

Dois endereços de ETDs remotos são considerados válidos:

ETD	Endereço
1	0
2	1

Isto é indicado no programa pela tabela etd_tab[].

Na nossa implementação consideramos que apenas o canal lógico 1 pode ser conectado, este é retornado ao processo chamador por meio de um apontador definido no programa como pcanal.

O passo 3 é implementado pela função de saída e o passo 4 pela função de próximo estado descritas anteriormente.

A função ESTABCV se encontra no arquivo estabcv.c.

ESTABCV utiliza a função de saída enviapl para enviar o caractere "CR".

4.2.1.2. Implementação de LIMPACV

LIMPACV inicia uma ação de desconexão de um CV; para isso ela envia um caractere de ESCAPE ao CX25 e muda o estado do CV para ESPPPTCLR (dependendo do estado anterior do CV e se o canal lógico for válido).

Esta função tem como parâmetro de entrada o número do canal lógico.

O algoritmo que representa LIMPACV é o seguinte:

1. Verificar se o canal lógico é inválido
2. Se sim, executar o passo 5
3. Caso contrário, enviar o caractere de ESCAPE ao CX25
4. Mudar o estado do CV
5. Retornar um código de retorno

Se o número do canal lógico for inválido o caractere de ESCAPE não será enviado e um código de retorno é retornado ao processo chamador indicando o ocorrido. Os passos 2 e 3 são realizados pelas funções de saída e de próximo estado respectivamente.

A função se encontra implementada no arquivo limpacv.c. LIMPACV utiliza a função de saida enviapl para enviar o caractere "ESC" ao CX25.

4.2.1.3. Implementação de RESETCV

A primitiva RESETCV dá inicio a um pedido de reinicialização do CV. Para tanto envia-se um caractere de ESCAPE para se entrar no modo de comando do CX25. O CV deve estar no estado DADOS ou N3OCP e o canal lógico deve ser válido.

Um código de retorno, indicando se a primitiva foi aceita ou a causa de rejeição, é retornado a ET.

O algoritmo da função RESETCV é apresentado logo abaixo:

1. Verificar se o canal lógico é inválido
2. Se sim, executar o passo 5
3. Caso contrário, enviar o caractere de ESCAPE para o CX25
4. Mudar o estado do CV
5. Enviar um código de retorno para ET

Os passos 2 e 3 são implementados pelas funções de saida e de próximo estado respectivamente.

A função RESETCV se encontra implementada no arquivo resetcv.c.

4.2.1.4. Implementação de TRAPACOTE

A função TRAPACOTE passa para a IX25 os dados para serem transmitidos pelo CV. TRAPACOTE fornece para a IX25 o número do canal lógico, o endereço do "buffer" onde estão localizados os dados (na implementação este endereço é dado pelo apontador para caracteres pbuffer) e o tamanho do "buffer" (fornecido pela variável tamanho). TRAPACOTE só enviará os dados se os parâmetros acima forem válidos e o estado do CV for DADOS. Caso contrário, um código de retorno é enviado ao processo chamador.

O algoritmo para TRAPACOTE é o seguinte:

1. Verificar se o canal lógico é inválido e se o endereço do "buffer" é inválido e se o tamanho do pacote é inválido
2. Se sim, executar o passo 5
3. Caso contrário, enviar os dados
4. Mudar de estado
5. Devolver um código de retorno

Os passos 4 e 5 são realizados pelas funções de saída e de próximo estado respectivamente.

A função TRAPACOTE utiliza a função de saída envia dados para enviar os dados, os quais são transformados antes de serem enviados e dependendo do tamanho do "buffer" de dados passado pela ET estes poderão não ser transferidos num só pacote (voltaremos ao assunto quando descrevermos envia dados). TRAPACOTE se encontra no arquivo trapacote.c.

4.2.1.5. Implementação de RCPACOTE

A função RCPACOTE transfere os dados da IX25 para a ET, fornece ainda o número de "bytes" transferidos e o número do canal lógico. Como uma função em C não pode retornar mais de um parâmetro, os dois últimos são passados por meio de um apontador para uma estrutura do tipo rcpacres, definida no arquivo cabeçalho ix25.h, assim descrita:

```
struct rcpacres {
     int nbtransf;  /* numero de bytes transferidos */
     int canal;
};
```

Para que RCPACOTE seja executada o endereço do "buffer" de recepção de dados da ET tem que ser válido, existir dados para serem transferidos e o CV estar num dos três estados ESPPPTDAD, DADOS ou N3OCP.

RCPACOTE não provoca nenhuma mudança de estado. O seu algoritmo é o seguinte:

1. Verificar se o endereço do "buffer" de recepção de dados da ET é válido e se há dados

2. Se não, executar o passo 4

3. Caso contrário, transferir os dados

4. Devolver um código de retorno

O passo 2 é realizado apenas olhando se a variável, definida no arquivo cabeçalho ix25global.h, tambufix25 (que fornece o tamanho do "buffer" de recepção de dados da IX25) é zero. O passo 3 é realizado pela função de saída. RCPACOTE utiliza a função de saída trans dados para realizar a transferência e se encontra no arquivo rcpacote.c.

4.2.1.6. Implementação de RCSINAL

RCSINAL tem a função de passar para a ET informações sobre o que está ocorrendo com o CV. Quatro tipos de informações são passadas:

1) CV encerrado - significa que o CV foi encerrado e deve estar disponível. No caso do ETD remoto ou a própria rede ter inicializado o encerramento, a causa e o diagnóstico desta é também passado para a ET.

2) CV reinicializado - significa que um "reset" foi dado no CV. Se foi o ETD remoto ou a própria rede que pediu a reinicialização, a causa e o diagnóstico desta é também passado para a ET.

3) CV estabelecido - significa que um CV foi estabelecido. Se o pedido foi realizado pelo ETD remoto, o endereço deste e uma informação adicional (não ainda implementada) são passadas para a ET.

4) Nada - como o próprio nome sugere, esta informação diz que não há nada a ser passado para ET.

Em qualquer caso o canal lógico é passado para a ET. Para se passar as complementações dos sinais acima quando da sinalização pelo ETD remoto ou pela rede, uma estrutura do tipo rcsinres, (definida no arquivo cabeçalho ix25.h), passada pela ET, é preeenchida e retornada para esta. A estrutura é como segue:

```
struct rcsinres {
        int s_canal;
        char causa[4];
        char diagnostico[4];
        char ber[4];    /* ender. etd rem. */
        char dur[17];   /* dados etd rem. */
};
```

RCSINAL pode ser chamada em qualquer estado do CV e não realiza nenhuma mudança no mesmo. O seu algoritmo é o seguinte:

1. Retornar o sinal e a estrutura de resposta para a ET

O único passo é implementado pela função de saída retsinal. RCSINAL se encontra no arquivo rcsinal.c.

4.2.2. Implementação das mensagens

Para tratarmos as mensagens que chegam do CX25, precisamos ter em mente como o sistema operacional UNIX faz operações de Entrada e Saída (E/S).

O UNIX utiliza o conceito de arquivo para se ter acesso aos mecanismos de E/S. Dessa forma quando quisermos fazer E/S é a mesma coisa que ler/gravar um/num arquivo. Esses arquivos são arquivos especiais no sentido de que eles não contêm informações e servem apenas para facilitar a manipulação de E/S [CHRI 85].

A simulação de recebimento de mensagens do CX25 é feita através do teclado do terminal de video.

Em linhas gerais o algoritmo de recepção de mensagens é o seguite:

1. Ler uma mensagem

2. Identificar a mensagem

3. Chamar a rotina de atendimento da mensagem

4. Ficar aguardando uma nova mensagem

Os passos 1 e 4 são implementados pela função lemsg e os passos 2 e 3 pela função idmsg que estão no arquivo rcmsg.c.

lemsg

A função lemsg fica sempre aguardando uma mensagem do CX25. Quando esta chega ela é lida no "buffer" de recepção de mensagens da IX25 bufmsg[], que é passado para a função de identificação de mensagens. É bom lembrar que qualquer mensagem termina com um "Carriage Return" seguido de um "Line Feed", na nossa implementação representados pelos caracteres "r" e "n" respectivamente, para efeito de visualização. Os caracteres de fim de mensagem são substituidos pelo caractere nulo (\0) no "buffer" para ficar compatível com a definição de um "string" em C.

idmsg

idmsg identifica e chama a rotina de tratamento da mensagem lida por lemsg.

Para realizarmos esta função dividimos as mensagens em três tipos:

1) Dados

2) Mensagens sem parâmetros

3) Mensagens com parâmetros

Os dados são identificados devido a peculiaridade com que são enviados. Se o primeiro caractere do "buffer" estiver entre "@" e "O" ou for um CR, então temos um "buffer" com dados, e chamamos a função de tratamento de dados.

No caso das mensagens sem parâmetros, nós organizamos estas mensagens numa tabela *tabmsg[] e fazemos uma pesquisa sequêncial para identificarmos a mensagem e chamarmos a sua respectiva rotina de tratamento.

As mensagens com parâmetros são identificadas pelo segundo caractere do "buffer", pois o primeiro é sempre '\', e daí chamamos a sua rotina de tratamento. Neste caso, cabe à rotina de tratamento a identificação correta da mensagem.

Uma mensagem de erro é retornada a lemsg no caso de haver alguma irregularidade.

As mensagens são totalmente descritas pelas tabelas de próximo estado e de funções de saída. Apresentaremos, portanto, uma breve discussão de como as mensagens estão situadas nas tabelas.

PPTDAD

Um "prompt" de dados só deve ser recebido quando a IX25 estiver no estado ESPPPTDAD. Esta mensagem habilita a IX25 a enviar dados para o CX25, mudando seu estado para DADOS no caso do "buffer" de transmissão de dados da IX25 estiver vazio, caso contrário, os dados restantes no referido "buffer" serão enviados e o próximo estado continuará sendo ESPPPTDAD. Isto é realizado pela função de saída pptdados. Se a mensagem for recebida em um outro estado, uma mensagem de erro é enviada.

PPTCMD

Quando um "prompt" de comandos chega a IX25 e esta etiver num dos estados seguintes ESPPPTSEL, ESPPPTRST ou ESPPPTCLR, o comando SEL, RESET ou CLR respectivamente, é enviado ao CX25 utilizando a função de saída enviapl. Se o estado for diferente dos estados acima, uma mensagem de erro é indicada.

FE

Quando um "FORMAT EFFECTOR" é recebido, ele confirma um comando de reinicialização enviado anteriormente. Portanto o CV deve estar no estado SNDRSTCMD e entrar para o estado ESPPPTDAD. Um sinal dizendo que a reinicialização foi efetuada é guardado no "buffer" de sinalização, utilizando-se a função de saída sinalpl para isso. Se o CV estiver em um outro estado uma mensagem de erro é indicada.

COM

A mensagem COM confirma um pedido de conexão realizado anteriormente, e só deve ser recebida se o CV estiver no estado SNDESTAB, o qual deve ser mudado para ESPPPTDAD. Uma sinalização indicando o ocorrido é guardada utilizando-se a função de saída sinalpl. Se o CV estiver num outro estado uma mensagem de erro será indicada.

CLRCONF

Um CLR CONF é recebido para indicar a confiramção de um pedido de desconexão enviado ao CX25 anteriormente. O CV deve estar no estado SNDENCRR e muda para DISPONIVEL após o recebimento da mensagem. Um sinal é guardado no "buffer" de sinalização indicando o fato, utilizando a função de saída sinalpl. Se o CV estiver em um outro estado, uma mensagem de erro é indicada.

OCUPADO

O recebimento da mensagem NIVEL 3 OCUPADO indica que o CX25 não pode mais receber dados da IX25 para transmitir ao ETD remoto. Esta mensagem só pode ser recebida se o CV estiver nos seguintes estados ESPPPTDAD ou DADOS. O CV passará para o estado N3OCP. Se o CV estiver em algum outro estado uma mensagem de erro será indicada.

DESOCUPADO

A mensagem NIVEL 3 DESOCUPADO informa a IX25 que o CX25 já está podendo enviar dados. Esta mensagem é recebida com o CV no estado N3OCP e muda este para ESPPPTDAD. Em qualquer outro estado uma mensagem de erro é indicada.

ERR1

Um ERR 1 é recebido para indicar um erro de sintaxe no comando enviado. Esta mensagem provoca uma repetição do comando enviado e só deve ser recebida nos estados SNDESTAB, SNDRSTCMD ou SNDENCRR que mudam para os estados ESPPPTSEL, ESPPPTRST ou ESPPPTCLR respectivamente. A função de saída enviapl é utilizada para enviar o caractere de escape. Se o CV estiver em um outro estado, uma mensagem de erro é indicada.

ERR2

Um ERR 2 é recebido para indicar um pedido de conexão num CV já estabelecido. Esta mensagem nunca vai ocorrer pela estrutura da nossa implementação.

ERR3

Uma mensagem ERR 3 indica que um pedido de desconexão foi feito num CV desconectado. Isto só pode ocorrer quando o CV estiver no estado SNDENCRR. Neste caso, o CV passa para o estado DISPONIVEL. Se esta mensagem ocorrer em um outro estado, uma mensagem de erro é indicada.

ATENCAO

Uma mensagem ATENCAO <BER> é recebida para indicar um pedido de conexão vindo do ETD remoto. A IX25 guarda o endereço do ETD remoto e um sinal indicando o ocorrido no "buffer" de sinalização. Consideramos apenas o caso do ETD remoto enviar 3 caracteres de endereço. A função de saída conind é utilizada para a execução da rotina. O CV deve estar no estado DISPONIVEL, que deve ser mudado para ESPPPTDAD. Se o CV estiver em algum outro estado, uma mensagem de erro é indicada.

RESET

Um RESET <causa> <diagnostico> é recebido para indicar um pedido de reinicialização vindo do ETD remoto ou da rede. A IX25 limpa seus "buffers" e guarda os códigos de causa e diagnóstico e o sinal indicando o ocorrido no "buffer" de sinalização. A função de saida rstind é utilizada para realizar a rotina. O CV deve estar em um dos seguintes estados ESPPPTDAD, DADOS, N3OCP, ESPPPTRST ou SNDRSTCMD e muda para o estado ESPPPTDAD. Se o CV estiver em algum outro estado uma mensagem de erro é indicada.

CLR

Uma mensagem CLR <causa> <disgnostico> é recebida para indicar um pedido de desconexão vindo do ETD remoto ou da rede. Os "buffers" da IX25 são inicializados e o "buffer" de sinalização é preenchido com os códigos de causa e diagnóstico e com o sinal indicando o ocorrido. A função de saida clrind é utilizada para executar a tarefa. O CV pode estar em qualquer estado exceto no estado DISPONIVEL no qual uma mensagem de erro é indicada. O estado do CV é mudado para DISPONIVEL em qualquer caso.

MDADOS

Quando recebemos dados do ETD remoto estes estão mascarados e devem voltar a forma original antes de serem colocados no "buffer" de recebimento de dados da IX25. A

função de saída rcdados é utilizada para a execução desta rotina. O CV deve estar em um dos seguintes estados, ESPPPTDAD, DADOS ou N3OCP. Caso contrário, uma mensagem de erro é indicada.

4.2.3. Implementação das Funções de Saída

São as seguintes, as funções definidas na tabela de saída:

1. retornapl
2. enviapl
3. envia_dados
4. pptdados
5. trans_dados
6. retsinal
7. sinalpl
8. conind
9. rstind
10. clrind
11. rcdados

Estas funções serão discutidas agora.

retornapl

retornapl é uma função bem simples, apenas retorna o parâmetro pl, o segundo membro da estrutura do tipo saida definida no arquivo cabeçalho ix25.h.

enviapl

enviapl envia um comando para o CX25 definido pelo parâmetro pl da estrutura do tipo saida. Os seguintes comandos são enviados:

1. ESCAPE - um caractere de ESCAPE é enviado para se entrar no modo de comandos.

2. CR - um caractere de retorno do carro é enviado para se entrar no modo de comandos quando do inicio de uma conexão de um CV.

3. "CLR"CR - os caracteres C, L, R e CR são enviados para informar o CX25 de um pedido de desconexão de um CV.

4. "RESETCV"CR - os caracteres R, E, S, E, T e CR são enviados para fornecer um comando de reinicialização do CV.

5. "SEL:"<BER>CR - os caracteres S, E, L, :, o endereço do ETD remoto e o caractere CR, são enviados para se fazer um pedido de conexão. O endereço do ETD remoto é dado por 3 caracteres

definidos por um "array" de apontadores para uma sequência de caracteres end etd[].

Os 4 primeiros comandos foram colocados numa tabela, *cmds[], na qual o parâmetro p1 é diretamente associado com uma entrada na tabela, facilitando a implementação.

O comando número 5 não poderia estar na tabela pois precisa de mais um parâmetro, o endereço do ETD remoto fornecido por e etd, definido no arquivo cabeçalho ix25global.h, que é diretamente associado a tabela end etd[].

Um código de retorno indica o sucesso da operação.

envia dados

A função de saída envia dados é implementada pensando-se em enviar os dados para o arquivo cmd x25.

O nosso problema é que não podemos enviar os dados entregues pela ET do modo que chegaram pois alguns caracteres tem um significado especial para o CX25, precisando serem de alguma forma mascarados.

Isso faz com que a IX25 não seja apenas uma implementação local, já que a interface remota tem que saber que modificações foram feitas nos dados para poder restaurar para o original e passá-los para a ET.

São os seguintes os caracteres que não podem aparecer

nos dados:

1. \ - este caractere não deve aparecer no inicio de uma linha de dados, para que não seja interpretado na recepção como um sinal de serviço.

2. * - este caractere também não deve aparecer no inicio de uma linha de dados, pois seria interpretado como sendo um "prompt" de dados.

3. > - este caractere, como os dois caracteres anteriores, não deve aparecer no inicio de uma linha de dados, pois seria interpretado como sendo um "prompt" de comandos.

4. CR - este caractere é interpretado como fim de mensagem e não pode aparecer em lugar algum nos dados.

5. ESCAPE - este caractere, como o caractere CR, não deve aparecer em lugar algum, para que não seja interpretado como um pedido para se sair do modo de transferência de dados para o modo de comandos.

Duas soluções foram estudadas para o problema:

1) A primeira solução consiste em redefinir os caracteres especiais de forma a perderem suas caracteristicas para o CX25. Isto é feito da seguinte forma:

a) Quando forem encontrados, no inicio de uma linha de dados, os caracteres "\", "*" ou ">", um caractere "%" será colocado antes de cada um dos caracteres acima, formando os pares "%\", "%*", "%>" respectivamente.

b) Os caracteres de retorno do carro e de ESCAPE serão substituídos por "%R" e "%E" respectivamente.

c) O caractere "%" será substituido por "%%".

Uma implicação desta solução é que se os dados passados pela ET tiverem o tamanho máximo de um pacote e alguns caracteres especiais aparecerem, então os dados não poderão sèr enviados num único pacote. Em vista disso temos que avisar ao receptor que o resto da mensagem ainda será enviada. Isto é realizado colocando-se o caractere "%" como o último caractere do pacote, e se o último caractere do pacote for "%" este será novamente enviado no próximo pacote. Um outro problema surge se o penúltimo caractere for um caractere especial e não for o último caractere da mensagem. Nesse caso, se o caractere especial for "CR", "ESC", "%", será enviado "%%" no primeiro pacote e "R", "E" e "%" no segundo pacote respectivamente.

2) A segunda solução consiste em mascarar todos os caracteres de dados indiscriminadamente da ET, de forma que os caracteres especias nunca apareçam. Isto é conseguido dividindo-se cada byte de dados em dois meio bytes e acrescentando-se à esquerda de cada um deles meio "byte" de PAD (que escolhemos como sendo 0100), como mostramos abaixo:

byte de dados

0 1 0 1	1 0 1 0
A	B

primeiro byte enviado

0 1 0 0	1 0 1 0
PAD	B

segundo byte enviado

0 1 0 0	0 1 0 1
PAD	A

Desse modo, apenas os caracteres ASCII @, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, "CR" e "LF" são enviados para o ETD remoto.

Vale a pena notar que para enviarmos mais de um pacote, o pacote de continuação deve esperar um "prompt" de dados antes de ser enviado. Vide a descrição pptdados adiante.

O fim da mensagem é indicado da seguinte forma, se a mensagem for menor que a metade do tamanho máximo de um pacote menos 1 (TAMMAXPAC/2 -1) este é indicado por um único FORMAT EFFECTOR, caso contrário, um FE será enviado no final do primeiro pacote e um outro no final da mensagem.

A primeira solução foi descartada por ter algoritmo muito maior e mais lento que a solução 2. Deve-se notar, entretanto que a solução 1 tem utilização melhor do CV para o caso em que os caracteres especiais aparecem num pacote de dados muito infrequentemente. A solução 2 leva a uma maior utilização do CV para o caso dos pacotes serem sempre maiores que TAMMAXPAC/2-1, mas tem a vantagem de ser bem mais simples para implementação. Esta solução foi adotada em comum acordo com a UFPe.

pptdados

pptdados trata a recepção de um "prompt" de dados quando a MEF está no estado ESPPPTDAD. Se o "buffer" de transmissão de dados da IX25 estiver vazio, a rotina apenas muda o estado da MEF habilitando a ET a enviar dados. Caso contrário, o pacote de continuação dos dados é mascarado e

enviado como definido em envia dados. O "buffer" de transmissão de dados é então esvaziado.

trans dados

Esta função faz a transferência dos dados do "buffer" da IX25 encontrado no "array" bufix25[], no arquivo cabeçalho ix25global.h, para o "buffer" passado pela ET. Após a cópia dos "buffers", são passados para a ET o número de bytes transferidos e o número do canal lógico. No final, o bufix25[] é zerado e um código de retorno dizendo que a operação foi bem sucedida é devolvido.

retsinal

retsinal tem a função de copiar o "buffer" de sinalização da IX25, que é uma estrutura do tipo rcsinres, para o "buffer" da ET. Após a cópia, limpar o "buffer" de sinalização da IX25. O sinal de retorno é dado pela variável s sinal, definida no arquivo cabeçalho ix25global.h.

sinalpl

sinalpl guarda o sinal dado pelo parâmetro pl, na variável s sinal do "buffer" de sinalizaçào da IX25. Isto ocorre quando uma confirmação de um comando enviado ao CX25 chega.

conind

Esta função trata uma indicação de conexão. A mensagem é verificada se está correta e então o campo de endereço remoto é copiado no "buffer" de sinalização juntamente com um sinal indicando o estabelecimento do CV.

rstind

Esta funcão verifica se a mensagem de RESET está correta e então inicializa todos as variáveis da IX25. Os campos de causa e diagnóstico são copiados no "buffer" de sinalização juntamente com o sinal indicando a realização de uma reinicialização.

clrind

Esta função verifica se a mensagem indicando um pedido de desconexão do lado remoto está correta. Os "buffers" internos da IX25 são inicializados. Os campos de causa e diagnóstico são copiados no "buffer" de sinalização juntamente com o sinal indicando que houve um encerramento do CV.

rcdados

Esta função realiza o tratamento dos dados enviados pelo ETD remoto. Cada caractere do "buffer" é verificado se está correto e então o meio "byte" de PAD é retirado e

combinado com o caractere seguinte para formar o "byte" de dados enviado pela ET. Este é então guardado no "buffer" de recepção de dados da IX25, bufix25[]. O final dos dados é verificado de acordo com o algoritmo definido na descrição da função de saída envia dados. Caso este seja detectado o tamanho do "buffer" de recepção de dados é copiado na variável tambufix25, para indicar que existe dados na IX25. Caso contrário, espera-se a chegada do pacote de continuação dos dados.

4.3. Comentários

Durante a implementação foi que sentimos melhor o que a IX25 deveria fazer. A dificuldade no desenvolvimento de um "software" desse tipo está no entrelaçamento das diversas funções, sempre que íamos implementando uma nova função tínhamos que ver o que seria afetado anteriormente, assim redefinimos varias vezes a nossa MEF. Paramos neste ponto a implementação mas sabemos que ela ainda pode ser melhorada, com a inclusão de novas funções como o envio de interrupção, "restart", "timeout", etc.

O código gerado está na biblioteca libix25.a e tem 12968 bytes. O tamanho do código fonte é mostrado na tabela abaixo:

LINHAS	PALAVR	CARAC	NOME
194	481	2985	ix25.h
535	1001	8353	ix25global.h
59	183	1075	estabcv.c
176	406	2569	f_saida.c
252	621	4204	f_saidal.c
35	74	513	limpacv.c
129	349	2471	rcmsg.c
23	55	449	rcpacote.c
15	31	232	rcsinal.c
22	43	351	resetcv.c
49	121	858	trapacote.c
1489	3365	24060	total

Supondo-se uma produção tipica de 600 linhas/mês, o esforço de programação levaria aproximadamente 3 homens-meses.

CAPÍTULO 5: TESTES

Na implementação de um protocolo, erros de lógica poderão ocorrer produzindo resultados inesperados. Assim, é necessário que a implementação seja testada para sabermos se conforma com a especificação do protocolo.

Inicialmente apenas testes individuais com cada função foram feitos ou seja, sem interação com qualquer outra função. Neste ponto o nosso "software" já poderia ser considerado executável, mas como podiam existir ainda erros de lógica, testes mais rigorosos foram realizados.

Idealmente os testes de um protocolo devem ser feitos na rede onde a implementação rodará. Quando a rede não está disponivel, é necessário uma simulação das condições de operação [SAUV 84a]. Para protocolos especificados usando modelos de MEF vários métodos podem ser aplicados para seleção de sequências de testes. Temos o método W, também chamado de Sequência de Caracterização, o método D, chamado de Sequência de Verificação e o método "Tournée de Transições" [SARI 84]. Dos métodos acima, o método "Tournée de Transições" pode deixar de detetar erros na função de próximo estado. Os outros dois métodos são completos na deteção de falhas, mas são mais complexos na geração das sequências de testes. Devido a simplicidade do método "Tournée de Transições", este foi escolhido para a realização dos testes com a IX25.

O método "Tournée de Transições" consiste em disparar todas as transições entre estados pelo menos uma vez, a partir do estado inicial da MEF.

A dificuldade na realização dos testes está no grande número de entradas (quando consideramos os parâmetros das funções), pois dependendo da combinação delas, a resposta da MEF poderá ser diferente. A seguinte sequência de testes foi executada manualmente:

```
/* Pedido de conexão recusado */
ESTABCV, PPTCMD, MCLR, RCSINAL
/* Indicação de conexão recusada */
ATENCAO, RCSINAL, LIMPACV, PPTCMD, CLRCONF, RCSINAL
/* Pedido de conexão com erro de sintaxe */
ESTABCV, PPTCMD, ERR1, PPTCMD, COM, RCSINAL
/* Pedido de desconexão com erro de sintaxe */
LIMPACV, PPTCMD, ERR1, PPTCMD, CLRCONF, RCSINAL
/* Indicação de conexão aceito */
ATENCAO, RCSINAL, PPTDAD
/* Envio de dados menor que TAMMAXPAC/2 */
TRAPACOTE, PPTDAD
/* Indicação de reinicialização */
RESET, RCSINAL, PPTDAD
/* Envio de dados maior que TAMMAXPAC/2 */
TRAPACOTE, PPTDAD, PPTDAD
/* Envio de dados igual a TAMMAXPAC/2 */
```

```
TRAPACOTE, PPTDAD, PPTDAD
/* Pedido de reinicialização com erro de sintaxe */
RESETCV, PPTCMD, ERR1, PPTCMD, FE, RCSINAL, PPTDAD
/* Envio de dados maior que TAMMAXPAC/2 com indicação
de ocupado */
TRAPACOTE, OCUPADO
/* Indicação de desocupado */
DESOCUPADO, PPTDAD, PPTDAD
/* Recepção de dados menor que TAMMAXPAC/2 */
MDADOS, RCPACOTE
/* Recepção de dados maior ou igual que TAMMAXPAC/2 */
MDADOS, RCPACOTE, MDADOS, RCPACOTE
/* Pedido de desconexão com indicação de ERR3 */
LIMPACV, PPTCMD, ERR3
/* Pedido de conexão aceito */
ESTABCV, PPTCMD, COM, RCSINAL, PPTDAD
/* Envio e Recepção de dados maior que TAMMAXPAC/2 */
TRAPACOTE, MDADOS
/* Indicação de reinicialização */
RESET, RCSINAL, PPTDAD
/* Indicação de ocupado */
OCUPADO
/* Recepção de dados */
MDADOS
/* Pedido de reinicialização */
RESETCV, PPTCMD, FE, RCSINAL
/* Pedido de desconexão com recepção de dados */
```

LIMPACV, MDADOS, PPTCMD, CLRCONF

Acreditamos que a sequência acima é suficiente para validar a IX25. Um algoritmo para geração automática da sequência de testes poderia seguir os passos abaixo:

1. Definir todas as transições possíveis a partir de cada estado.

2. Definir um estado inicial. (Sugestão: De preferência um estado que seja mais fácil retornar a partir de qualquer outro estado.)

3. Definir um caminho de retorno ao estado inicial a partir de cada estado.

4. Definir um caminho para chegar a cada estado a partir do estado inicial.

5. Tomar um estado para pesquisa.

6. Há transições a partir do estado escolhido em 5, que não foram pesquisadas?
   Caso positivo:

   6.1. Executar uma das transições, marcando a transição executada.

   6.2. Voltar ao estado inicial, marcando as transições executadas.

6.3. Voltar ao estado pesquisado, marcando as transições executadas.

6.4. Voltar ao passo 6.

Caso Negativo: Vá para 7.

7. Há estados que ainda não foram pesquisados?

Caso Positivo:

7.1. Voltar ao estado inicial, marcando as transições executadas.

7.2. Ir para o estado seguinte, marcando as transições executadas.

7.3. Voltar ao passo 6.

Caso Negativo: Termine o algoritmo.

Nos testes acima, é interessante se ter uma maneira automática para chegarmos a um ponto na sequência onde um erro foi encontrado. Isto facilita muito a depuração. Evidente que a modificação do "software" pode produzir resultados inesperados em sequências já testadas. Assim, no momento em que um erro for reparado, toda a sequência deve ser novamente testada, para garantirmos a conformidade do "software", a menos das limitações do método escolhido.

CAPÍTULO 6: CONCLUSÃO

Era nossa intenção ao iniciar o projeto CEPINNE, fornecer os serviços básicos para que a rede pudesse entrar em funcionamento, ou seja, simplificar ao máximo para depois ir acrescentando os melhoramentos que se fizessem necessários.

Foi com esse pensamento que o presente trabalho foi realizado. Na primeira fase do projeto, foi estudado em detalhes, o protocolo de Transporte que seria utilizado, incluindo as primitivas que seriam passadas para a Interface X.25. Nessa fase, também decidiu-se o poderia ser simplificado, desse modo, não teríamos controle de "timeout", não teríamos interrupção, a Estação de Transporte não faria multiplexação, etc.

A segunda fase consistiu no estudo da documentação do Conversor X.25 que seria utilizado e nesse ponto foi que surgiram as maiores dificuldades, como mostrado no capítulo 3. Depois que especificamos a Interface, sentimos a necessidade de fazer uma validação da nossa especificação. Mas tínhamos que desenvolver toda uma ferramenta que não é muito simples, para realizar nosso intento. Fica aqui então a sugestão para um possível trabalho, para um curso de Protocolos, por exemplo, o desenvolvimento de um método de validação automática de protocolos. A seguinte bibliografia pode ser consultada [WEST 78], [BOCH 78], [ZAFI 78], [WEST 78a], [WEST 78b].

A última fase realizaria a implementação propriamente dita, seguida dos melhoramentos que se fizessem necessários a medida que utilizassemos a rede.

Conexão Real da IX25 com o CX25

Como já dissemos antes a comunicação do CX25 com o ETD local é feita através de uma interface RS232C/V.24 assincrona e portanto compativel com as interfaces seriais assincronas do PDP-11/34.

Sempre que um novo periférico é adicionado ou removido, o sistema operacional precisa ser modificado. O que temos a fazer é reconfigurar o sistema UNIX de modo a alocar uma interface serial do tipo RS232C para o CX25 e então criar um arquivo especial no diretório /dev utilizando o comando mknod para fazermos a ligação entre o CX25 e o sistema operacional [CHRI 85].

BIBLIOGRAFIA

[BOCH 78] Bochmann, G. V., Finite State Description of Communication Protocols, Computer Network, vol 12, october 1978.

[CEPI PTR] Projeto Técnico da Rede CEPINNE, Vol 1 e 2, Embratel.

[CHAP 83] Chapin, A. L., Computer Communication Standards, Computer Communication Review, vol 13, número 1, January 1983.

[CHRI 85] Christian, K., Sistema Operacional Unix, Editora Campus, 1985.

[COHE 78] Cohen, N. B., Implementation and Performance of an X.25 Packet Network Interface Machine, Department of Computer Science and Computer Communications Networks Group, Universidade de Waterloo, September 1978.

[CX25 ED] Especificação de Definição do Conversor X.25, CPqD/TELEBRAS.

[CX25 MO] Conversor X.25, Manual de Operação, Icatel

[CX25 MU] Manual de Utilização do Conversor X.25, CPqD/TELEBRAS, Julho/83.

[DANT 80] Danthine, A., Protocol Representation with Finite Models, IEEE Transactions on Communications, Vol. COM - 28, No. 4, April 1980, pp 632-643.

[DONO 72] Donovan, J. J., Systems Programming, McGraw-Hill, Inc., 1972 pp 239-240.

[KERN 78] Kernighan, B. W., Ritchie, D. M., The C Programming Language, Prentice-Hall inc., 1978.

[PUC 82] Especificação de um Protocolo de Transporte para Redes que Utilizam Circuitos Virtuais, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Dezembro de 1982.

[RENP 83] Especificação de Protocolos de Acesso à RENPAC (X.3, X.28, X.29, X.25), Departamento de Comunicação de Dados, Área de Desenvolvimento, EMBRATEL, Jun/83.

[SARI 84] Sarikaya, B., Bochmann, G. V., Synchronization and Specification Issues in Protocol Testing, IEEE Transactions on Communications, Vol. COM-32, NO. 4, April 1984.

[SAUV PAN] Sauvé, J. P., Uma Visão da Padronização Internacional de Protocolos de Alto Nível para Redes de Computadores, Departamento de Sistemas

e Computação, Grupo de Redes de Computadores, UFPb.

[SAUV 84] Sauvé J. P., Equidade em Redes de Computadores, Anais do II Simpósio Brasileiro sobre Redes de Computadores, Abril de 1984.

[SAUV 84a] Sauvé, J. P., Moura, J. A. B., Giozza, W. F., Araújo, J. F. M., Projeto e Desenvolvimento de Protocolos, GRC/UFPb, Novembro de 1984.

[SLOM 78] Sloman, M. S., X.25 Explained, Computer Communications IPS Publications, February 1978.

[UNIX 79] UNIX Programmer's Manual, Seven Edition, Vol. 2B, Bell Laboratories, January 1979.

[WEST 78] West, C. H., General Technique for Communication Protocol Validation, IBM J. Res. Develop., vol 22 número 4, july 1978.

[WEST 78a] West, C. H., An Automated Technique of Communications Protocol Validation, IEEE Transactions on Communications, Vol. COM-26, August 1978.

[WEST 78b] West, C. H., Zafiropulo, P., Automated Validation of a Communications Protocol: the CCITT X.21 Recommendation, IBM J. Res. Develop., Vol 22, No. 1, january 1978.

[ZAFI 78] Zafiropulo, P., Protocol Validation by Duologue-Matrix Analysis, IEEE Transactions on Communications, Vol. COM-26, August 1978.

APÊNDICE

```
/*
 * Definicoes para se interfacear com as rotinas do X25.
 */
#include        <stdio.h>

/*
 * Saidas da Interface X.25
 */

/*
 * Comandos
 */
#define ESCAPE  0
#define RETCAR  1
#define CLR     2
#define RST     3
#define SEL     4

/*
 * Codigos de retorno
 */

/*
 * retorno para todos
 */
#define OK              0

/*
 * retorno para estabcv
 */
#define EETDINVALIDO    1
#define ENAOHACV        2

/*
 * retorno para limpacv
 */
#define LCVINVALIDO     10
#define LPCMDN3         11
#define LCVNAOPRONTO    12

/*
 * retorno para resetcv
 */
#define RCVINVALIDO     20
#define RCVENCERRANDO   21
#define RCVRESETANDO    22
#define RPCMDN3         23
#define RCVNAOPRONTO    24

/*
 * retorno para trapacote
 */
#define TCVINVALIDO     40
#define TBUFINVALIDO    41
```

```
#define TTAMINVALIDO    42
#define TCVNAOPRONTO    43
#define TCVRESETANDO    44
#define TCVENCERRANDO   45
#define TN3OCUPADO      46
#define TESPPPTDAD      47
#define TERRODISP       48

/*
 * retorno para rcpacote
 */
#define RPBUFINVALIDO   50
#define RPNAOHADADOS    51
#define RPCVNAODISP     52
#define RPCVNAOPRONTO   53
#define RPCVRESETANDO   54
#define RPCVENCERRANDO  55

/*
 * sinalizacao para rcsinal
 */
#define X25_NADA                0
#define X25_ENCERRAMENTO        1
#define X25_RESET               2
#define X25_ESTABELECIMENTO     4

/*
 * estados do circuito virtual
 */
#define DISPONIVEL      0
#define ESPPPTSEL       1
#define SNDESTAB        2
#define ESPPPTDAD       3
#define DADOS           4
#define N3OCP           5
#define ESPPPTRST       6
#define SNDRSTCMD       7
#define ESPPPTCLR       8
#define SNDENCRR        9

/*
 * entradas da interface X.25
 */

/*
 * primitivas
 */
#define ESTABCV         0
#define LIMPACV         1
#define RESETCV         2
#define TRAPACOTE       3
#define RCPACOTE        4
#define RCSINAL         5
```

```
/*
 * mensagens
 */
#define PPTDAD          6
#define PPTCMD          7
#define FE              8
#define COM             9
#define CLRCONF         10
#define OCUPADO         11
#define DESOCUPADO      12
#define ERR1            13
#define ERR2            14
#define ERR3            15
#define ATENCAO         16
#define MCLR            17
#define RESET           18
#define MDADOS          19

/*
 * Definicoes para as funcoes conind, rstind, clrind
 */
#define SP      ' '
#define CMP_ATC 9
#define CMP_RST 7
#define CMP_CLR 5
#define CMP_BER 3
#define CMP_CAUSA       3
#define CMP_DIAG        3

/*
 * Definicoes para recepcao e envio de dados
 */
#define MASC    017             /* 00001111 */
#define PRIMEIRO        1
#define SEGUNDO 2
#define PAD     0100            /* 01000000 */
/*
 * indicacao de mensagem errada
 */
#define MSGERR  -1

/*
 * tamanho maximo de um pacote de dados
 */
#define TAMMAXPAC       20

/*
 * estrutura de resposta para rcpacote
 */
struct rcpacres {
        int nbtransf; /* numero de bytes transferidos */
        int canal;
};
```

```
/*
 * estrutura de resposta para rcsinal
 */
struct rcsinres {
        int s_canal;
        char causa[4];
        char diagnostico[4];
        char ber[4];            /* endereco do etd remoto */
        char dur[17];           /* dados do etd remoto */
};
/*
 * estrutura para tabela de saida
 */
struct saida {
        int (*func)();  /* apontador para uma funcao */
        int pl;
};

/*
 * Definicoes das funcoes de saida
 */
int enviapl();
int retornapl();
int envia_dados();
int pptdados();
int trans_dados();
int retsinal();
int sinalpl();
int conind();
int rstind();
int clrind();
int rcdados();
```

```
/*
 * variaveis compartilhadas por mais de uma rotina
 */

/*
 * Endereco do ETD remoto
 */
int e_etd;

/*
 * resposta para rcsinal
 */
struct rcsinres inircsin = {
        -1,
        '\0',
        '\0',
        '\0',
        '\0'
};
struct rcsinres *respsinal = &inircsin;
int s_sinal;    /* guarda sinal */

/*
 * Buffer de transmissao de dados
 */
char buftrans[TAMMAXPAC + 1];

/*
 * Tamanho do buffer de trans. de dados
 */
int tambuftrans;

/*
 * buffer de recepcao de dados da ix25
 */
char bufix25[TAMMAXPAC];

/*
 * tamanho do buffer de rec. de dados da ix25
 */
int tambufix25;

/*
 * variaveis para controlar recepcao de dados
 */
int pacote = PRIMEIRO;
int deslocamento;
/*
 * Estado presente do canal logico
 */
int epcanal = DISPONIVEL;

/*
 * Tabela de proximo estado
```

```
 */
int fpe [] [10] = {
{       /* ESTABCV */
        ESPPPTSEL,      /* DISPONIVEL */
        ESPPPTSEL,      /* ESPPPTSEL  */
        SNDESTAB,       /* SNDESTAB */
        ESPPPTDAD,      /* ESPPPTDAD */
        DADOS,          /* DADOS */
        N3OCP,          /* N3OCP */
        ESPPPTRST,      /* ESPPPTRST */
        SNDRSTCMD,      /* SNDRSTCMD */
        ESPPPTCLR,      /* ESPPPTCLR */
        SNDENCRR        /* SNDENCRR */
},
{       /* LIMPACV */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTCLR,
        ESPPPTCLR,
        ESPPPTCLR,
        ESPPPTCLR,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* RESETCV */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        ESPPPTRST,
        ESPPPTRST,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* TRAPACOTE */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        ESPPPTDAD,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* RCPACOTE */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
```

```
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* RCSINAL */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* PPTDAD */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        DADOS,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* PPTCMD */
        DISPONIVEL,
        SNDESTAB,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        SNDRSTCMD,
        SNDRSTCMD,
        SNDENCRR,
        SNDENCRR
},
{       /* FE */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        ESPPPTDAD,
        ESPPPTCLR,
```

```
        SNDENCRR
},
{       /* COM */
        DISPONIVEL,
        ESPPPTSEL,
        ESPPPTDAD,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* CLR CONF */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        DISPONIVEL
},
{       /* OCUPADO */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        N3OCP,
        N3OCP,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* DESOCUPADO */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        ESPPPTDAD,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* ERR1 */
        DISPONIVEL,
        ESPPPTSEL,
        ESPPPTSEL,
```

```
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        ESPPPTRST,
        ESPPPTCLR,
        ESPPPTCLR
},
{       /* ERR2 */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* ERR3 */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        DISPONIVEL
},
{       /* ATENCAO */
        ESPPPTDAD,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* MCLR */
        DISPONIVEL,
        DISPONIVEL,
        DISPONIVEL,
        DISPONIVEL,
        DISPONIVEL,
        DISPONIVEL,
        DISPONIVEL,
        DISPONIVEL,
        DISPONIVEL,
```

```
        DISPONIVEL
},
{       /* RESET */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        ESPPPTDAD,
        ESPPPTDAD,
        ESPPPTDAD,
        ESPPPTDAD,
        ESPPPTCLR,
        SNDENCRR
},
{       /* MDADOS */
        DISPONIVEL,
        ESPPPTSEL,
        SNDESTAB,
        ESPPPTDAD,
        DADOS,
        N3OCP,
        ESPPPTRST,
        SNDRSTCMD,
        ESPPPTCLR,
        SNDENCRR
}
};

/*
 * Tabela das funcoes de saida
 */
struct saida fs [] [10] = {
{       /* ESTABCV */
        {enviapl, RETCAR},      /* DISPONIVEL */
        {retornapl, ENAOHACV},  /* ESPPPTSEL */
        {retornapl, ENAOHACV},  /* SNDESTAB */
        {retornapl, ENAOHACV},  /* ESPPPTDAD */
        {retornapl, ENAOHACV},  /* DADOS */
        {retornapl, ENAOHACV},  /* N3OCP */
        {retornapl, ENAOHACV},  /* ESPPPTRST */
        {retornapl, ENAOHACV},  /* SNDRSTCMD */
        {retornapl, ENAOHACV},  /* ESPPPTCLR */
        {retornapl, ENAOHACV}   /* SNDENCRR */
},
{       /* LIMPACV */
        {retornapl, LCVINVALIDO},
        {retornapl, LCVNAOPRONTO},
        {retornapl, LCVNAOPRONTO},
        {enviapl, ESCAPE},
        {enviapl, ESCAPE},
        {enviapl, ESCAPE},
        {enviapl, ESCAPE},
        {retornapl, LPCMDN3},
        {enviapl, ESCAPE},
```

```
        {retornapl, OK}
},
{       /* RESETCV */
        {retornapl, RCVINVALIDO},
        {retornapl, RCVNAOPRONTO},
        {retornapl, RCVNAOPRONTO},
        {retornapl, RPCMDN3},
        {enviapl, ESCAPE},
        {enviapl, ESCAPE},
        {retornapl, RPCMDN3},
        {retornapl, RCVRESETANDO},
        {retornapl, RPCMDN3},
        {retornapl, RCVENCERRANDO}
},
{       /* TRAPCOTE */
        {retornapl, TCVINVALIDO},
        {retornapl, TCVNAOPRONTO},
        {retornapl, TCVNAOPRONTO},
        {retornapl, TESPPPTDAD},
        {envia_dados, NULL},
        {retornapl, TN3OCUPADO},
        {retornapl, TCVRESETANDO},
        {retornapl, TCVRESETANDO},
        {retornapl, TCVENCERRANDO},
        {retornapl, TCVENCERRANDO}
},
{       /* RCPACOTE */
        {retornapl, RPCVNAODISP},
        {retornapl, RPCVNAOPRONTO},
        {retornapl, RPCVNAOPRONTO},
        {trans_dados, NULL},
        {trans_dados, NULL},
        {trans_dados, NULL},
        {retornapl, RPCVRESENTANDO},
        {retornapl, RPCVRESENTANDO},
        {retornapl, RPCVENCERRANDO},
        {retornapl, RPCVENCERRANDO}
},
{       /* RCSINAL */
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL},
        {retsinal, NULL}
},
{       /* PPTDAD */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
```

```
        {pptdados, OK},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
},
{       /* PPTCMD */
        {retornapl, MSGERR},
        {enviapl, SEL},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, RST},
        {retornapl, MSGERR},
        {enviapl, CLR},
        {enviapl, CLR}
},
{       /* FE */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {sinalpl, X25_RESET},
        {enviapl, ESCAPE},
        {retornapl, MSGERR},
},
{       /* COM */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {sinalpl, X25_ESTABELECIMENTO},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
},
{       /* CLR CONF */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {sinalpl, X25_ENCERRAMENTO}
```

```
},
{       /* OCUPADO */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, OK},
        {retornapl, OK},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
},
{       /* DESOCUPADO */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, OK},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
},
{       /* ERR1 */
        {retornapl, MSGERR},
        {enviapl, RETCAR},
        {enviapl, RETCAR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {enviapl, ESCAPE},
        {enviapl, ESCAPE}
},
{       /* ERR2 */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
},
{       /* ERR3 */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
```

```
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {sinalpl, X25_ENCERRAMENTO}
},
{       /* ATENCAO */
        {conind, OK},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
},
{       /* MCLR */
        {retornapl, MSGERR},
        {clrind, OK},
        {clrind, OK},
        {clrind, OK},
        {clrind, OK},
        {clrind, OK},
        {clrind, OK},
        {clrind, OK},
        {clrind, OK},
        {clrind, OK}
},
{       /* RESET */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {rstind, OK},
        {rstind, OK},
        {rstind, OK},
        {rstind, OK},
        {rstind, OK},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
},
{       /* MDADOS */
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {rcdados, OK},
        {rcdados, OK},
        {rcdados, OK},
        {retornapl, MSGERR},
        {retornapl, MSGERR},
        {enviapl, ESCAPE},
        {retornapl, MSGERR}
```

```
}
};
```

```
/*
 * estabelece um circuito virtual
 */
#include        "ix25.h"


extern epcanal;
extern e_etd;
extern struct rcsinres *respsinal;
extern struct saida fs [] [10];
extern int fpe [] [10];

estabcv(etd, pcanal)
int etd;        /* endereco do ETD remoto */
char *pcanal;   /* para retornar numero canal logico */
{
        int codret;     /* codigo de retorno */

        if(etd_invalido(etd)) {         /* etd remoto invalido? */
                return( EETDINVALIDO );
        }
        e_etd = etd;
        if((*pcanal = cv_livre()) < 0) {  /* can. log. disponivel? */
                return( ENAOHACV );
        }

        respsinal->s_canal = *pcanal;
        codret = (*fs[ESTABCV][epcanal].func)
                                (fs[ESTABCV][epcanal].pl);
        epcanal = fpe[ESTABCV][epcanal];
        return( codret );
}

/*
 * Funcoes de estabcv.c
 */

/*
 * Verifica o endereco do ETD remoto
 */
#define NETD    2       /* numero de ETDs na rede */

static int etd_tab [NETD] = {0, 1};     /* end. dos ETDs remotos */

etd_invalido (etd)
int etd;        /* endereco do ETD remoto */
{
        int i;

        for(i = 0; i <=NETD && etd != etd_tab[i]; i++)
                ;
        return(i > NETD);
}
```

estabcv.c

```
/*
 * Se o canal estiver ocupado retorna -1
 * ou o numero do canal logico
 */
cv_livre()
{
        return((epcanal == DISPONIVEL) ? 1 : -1);
}
```

```
/*
 * Encerra o circuito virtual
 */
#include        "ix25.h"

limpacv(canal)
int canal;      /* numero do canal logico */
{
        extern epcanal;
        extern struct saida fs [] [10];
        extern int fpe [] [10];
        int codret;

        if(canal_inv(canal)) {
                return( LCVINVALIDO );
        }

        codret = (*fs[LIMPACV][epcanal].func)
                        (fs[LIMPACV][epcanal].pl, NULL);
        epcanal = fpe[LIMPACV][epcanal];
        return(codret);
}

/*
 * Funcoes de limpacv
 */

/*
 * Verifica se canal logico e' invalido
 */

canal_inv(canal)
int canal;
{
        return( canal != 1 );
}
```

```
/*
 * Transmite um pacote de dados
 */
#include        "ix25.h"

trapacote(canal, pbuffer, tamanho)
int canal;      /* numero do canal logico */
char *pbuffer;  /* apontador para o buffer de dados */
unsigned tamanho;       /* numero de bytes do buf de dados */
{
        extern struct saida fs [] [10];
        extern int fpe [] [10];
        extern epcanal;
        extern tambuftrans;

        int codret;

        if(tambuftrans != 0) {
                return( TESPPPTDAD );
        }
        if(canal_inv(canal)) {
                return( TCVINVALIDO );
        }
        if(end_inv(pbuffer)) {
                return( TBUFINVALIDO );
        }
        if(tamanho > TAMMAXPAC) {
                return( TTAMINVALIDO );
        }

        codret = (*fs[TRAPACOTE][epcanal].func)
                        (fs[TRAPACOTE][epcanal].pl,tamanho, pbuffer);
        epcanal = fpe[TRAPACOTE][epcanal];
        return(codret);
}

/*
 * funcoes de TRAPACOTE
 */

/*
 * endereco do buffer de dados invalido
 */

end_inv(pbuffer)
char *pbuffer;
{
        return(pbuffer == NULL);
}
```

```
/*
 * Recebe um pacote de dados
 */
#include        "ix25.h"

rcpacote(bufrcpac, resposta)
char *bufrcpac;         /* buffer de recepcao */
struct rcpacres *resposta;
{
        extern struct saida fs [] [10];
        extern epcanal, tambufix25;
        int codret;

        if(end_inv(bufrcpac)) {
                return( RPBUFINVALIDO );
        }
        if(tambufix25 == 0) {
                return( RPNAOHADADOS );
        }
        codret = (*fs[RCPACOTE][epcanal].func)
                                (fs[RCPACOTE][epcanal].pl,
                                        bufrcpac, resposta);

        return(codret);
}
```

```
/*
 * Recebe sinal da ix25
 */
#include        "ix25.h"

rcsinal(resposta)
struct rcsinres *resposta;
{
        extern struct saida fs [] [10];
        extern epcanal;
        int sinret;

        sinret = (*fs[RCSINAL][epcanal].func)(resposta);
        return( sinret );
}
```

```
/*
 * Reinicializa o circuito virtual
 */

#include        "ix25.h"

resetcv(canal)
int canal;
{
        extern epcanal;
        extern struct saida fs [] [10];
        extern int fpe [] [10];
        int codret;

        if(canal_inv(canal)) {
                return( RCVINVALIDO );
        }

        codret = (*fs[RESETCV][epcanal].func)(fs[RESETCV][epcanal].pl);
        epcanal = fpe[RESETCV][epcanal];
        return (codret);
}
```

```
/*
 * Funcoes da tabela de saida
 */
#include        "ix25.h"

extern char buftrans[];
extern int tambuftrans;
/*
 * Retorna o parametro pl
 */
retornapl(pl)
int pl;
{
        return(pl);
}

/*
 * Envia comando para o CX25
 */

char * cmds [] = {
        {"'ESC'"},
        {"'CR'"},
        {"CLR'CR'"},
        {"RESET'CR'"}
};

char *end_etd [] = {"000", "001"};

enviapl (pl)
int pl;
{
        FILE *fp;
        extern int e_etd;

        if((fp = fopen("cmd_x25", "w")) == NULL) {
                fprintf(stderr, "nao pode abrir cmd_x25\n");
                return( ENAOHACV );
        }
        if(pl == SEL) {
                fprintf(fp, "SEL:%s'CR'", end_etd[e_etd]);
        }
        else {
                fprintf(fp, "%s", cmds[pl]);
        }
        fclose(fp);
        return( OK );
}

/*
 * Envia dados para o CX25
 */

envia_dados(nulo, tamanho, pbuffer)
```

```
int nulo;
unsigned tamanho;
char *pbuffer;
{
        char *apb;
        int nbytes;     /* cont. numero de bytes  de dados */

        apb = buftrans;
        for(nbytes = 1; nbytes <= tamanho; nbytes++) {
                *apb++ = *pbuffer++;
        }
        *apb = '\0';
        for(nbytes=1, apb=buftrans; nbytes<=tamanho &&
                                nbytes<=TAMMAXPAC/2; nbytes++) {
                if(muda_dados(*apb++) == TERRODISP) {
                        return( TERRODISP );
                }
        }
        if(tamanho < TAMMAXPAC/2) {
                if(manda('r') == TERRODISP) {
                        return( TERRODISP );
                }
                tambuftrans = 0;
        } else {
                tambuftrans = TAMMAXPAC/2;
        }
        return( OK );
}
/*
 * funcoes de envia dados
 */

/*
 * transforma os dados antes de enviar
 */

muda_dados(c1)
char c1;
{
        char c2;

        c2 = c1;

        c1 = (c1 & MASC) | PAD;
        if(manda(c1) == TERRODISP) {
                return( TERRODISP );
        }

        c2 = (MASC & (c2 >> 4)) | PAD;
        if(manda(c2) == TERRODISP) {
                return( TERRODISP );
        }
        return( OK );
}
```

f_saida.c

```
/*
 * manda um caractere para CX25
 */

manda(c)
char c;
{
        FILE *fp;

        if((fp = fopen("cmd_x25", "a")) == NULL) {
                fprintf(stderr, "nao pode abrir cmd_x25\n");
                return( TERRODISP );
        }
        fprintf(fp, "%c", c);
        fclose(fp);
        return( OK );
}

/*
 * trata a recepcao de um prompt de dados
 */

pptdados()
{
        char *apb;
        int nbytes;

        if(tambuftrans == 0) {
                return( OK );
        } else {
                apb = buftrans + tambuftrans;
                for(nbytes=1; nbytes<=TAMMAXPAC/2 && *apb!='\0';
                                                        nbytes++) {
                        if(muda_dados(*apb++) == TERRODISP) {
                                return( MSGERR );
                        }
                }
                if(nbytes < TAMMAXPAC/2) {
                        if(manda('r') == TERRODISP) {
                                return( MSGERR );
                        }
                }
                return( OK );
        }
}

/*
 * Tranfere dados da ix25 para a ET
 */

trans_dados(nulo, bufrcpac, resposta)
int nulo;
char *bufrcpac;
struct rcpacres *resposta;
```

f_saida.c

```
{
        extern char bufix25[];
        extern tambufix25;
        int i;

        for(i = 0; i < tambufix25; i++) {
                *bufrcpac++ = bufix25[i];
        }
        *bufrcpac = '\0';
        resposta->nbtransf = tambufix25;
        resposta->canal = 1;

        tambufix25 = 0;
        return( OK );
}
```

```
/*
 * Continuacao das funcoes de saida
 */
#include        "ix25.h"

extern struct rcsinres *respsinal;
extern int s_sinal;
extern int tambufix25;
extern char bufix25[];
extern int pacote;
extern int deslocamento;

/*
 * Retorna sinalizacao para ET
 */

retsinal(resposta)
struct rcsinres *resposta;
{
        int sinret;

        resposta->s_canal = respsinal->s_canal;
        strcpy(resposta->causa, respsinal->causa);
        strcpy(resposta->diagnostico, respsinal->diagnostico);
        strcpy(resposta->ber, respsinal->ber);
        strcpy(resposta->dur, respsinal->dur);

        sinret = s_sinal;

        limpasinal();
        return( sinret );
}

/*
 * Sinaliza a confirmacao de um comando
 */

sinalpl(pl)
int pl;
{
        limpasinal();
        s_sinal = pl;

        return( OK );
}

/*
 * Trata uma indicacao de conexao
 */
conind(nulo, pbuf)
int nulo;
char *pbuf;
{
        static char at[] = "\\ATENCAO ";
```

```
        if(strncmp(at, pbuf, CMP_ATC) != 0) {
                limpasinal();
                return( MSGERR );
        } else if(strlen(pbuf + CMP_ATC) != CMP_BER) {
                limpasinal();
                return( MSGERR );
        } else {
                limpasinal();
                respsinal->s_canal = 1;
                strcpy(respsinal->ber, (pbuf + CMP_ATC));
                s_sinal = X25_ESTABELECIMENTO;
                return( OK );
        }
}

/*
 * Trata uma indicacao de desconexao
 */
clrind(nulo, pbuf)
int nulo;
char *pbuf;
{
        static char clr[] = "\\CLR ";
        int desl;       /* deslocamento no campo de causa */

        if(strncmp(clr, pbuf, CMP_CLR) != 0) {
                return( MSGERR );
        }
        limpasinal();
        if((desl = cpcausa(pbuf+CMP_CLR)) > CMP_CAUSA) {
                return( MSGERR );
        }
        if(cpdiag(pbuf+CMP_CLR+desl+1) > CMP_DIAG) {
                return( MSGERR );
        }
        limpabuf();
        s_sinal = X25_ENCERRAMENTO;
        return( OK );
}

/*
 * Trata uma indicacao de reinicializacao
 */
rstind(nulo, pbuf)
int nulo;
char *pbuf;
{
        static char rst[] = "\\RESET ";
        int desl;       /* deslocamento no campo de causa */

        if(strncmp(rst, pbuf, CMP_RST) != 0) {
                return( MSGERR );
        }
        limpasinal();
```

```
        if((desl = cpcausa(pbuf+CMP_RST)) > CMP_CAUSA) {
                return( MSGERR );
        }
        if(cpdiag(pbuf+CMP_RST+desl+1) > CMP_DIAG) {
                return( MSGERR );
        }
        limpabuf();
        s_sinal = X25_RESET;
        return( OK );
}

/*
 * Recepcao de dados do ETD remoto
 */
rcdados(nulo, pbuf)
int nulo;
char *pbuf;
{
        int ord_caract;
        char c, ctmp, *ap;

/* erro no primeiro pacote */
        if(pacote == SEGUNDO && deslocamento == 0) {
                pacote = PRIMEIRO;
                return( MSGERR );
        }
        ap = bufix25 + deslocamento;
        ord_caract = PRIMEIRO;
        while((c = *pbuf++) != '\0') {
                if(c <= '@' && c >= '0') {
                        ord_caract = SEGUNDO;
                        break;
                } else if(ord_caract == PRIMEIRO) {
                        ctmp = c & MASC;
                        ord_caract = SEGUNDO;
                        continue;
                } else if(ord_caract == SEGUNDO) {
                        c &= MASC;
                        *ap++ = ctmp | (c << 4);
                        ord_caract = PRIMEIRO;
                        continue;
                }
        }
        if(ord_caract == SEGUNDO) {
                bufix25[0] = '\0';
                deslocamento = 0;
                if(pacote == PRIMEIRO) {
                        pacote = SEGUNDO;
                } else {
                        pacote = PRIMEIRO;
                }
                return( MSGERR );
        }
        *ap = '\0';
```

```
        if(pacote == PRIMEIRO) {
                if((tambufix25 = strlen(bufix25)) < (TAMMAXPAC/2 - 1)) {
                        deslocamento = 0;
                        return( OK );
                } else if(tambufix25 == TAMMAXPAC/2) {
                        pacote = SEGUNDO;
                        deslocamento = tambufix25;
                        tambufix25 = 0;
                        return( OK );
                } else {
                        pacote = SEGUNDO;
                        deslocamento = 0;
                        tambufix25 = 0;
                        return( MSGERR );
                }
        } else {
                tambufix25 = strlen(bufix25);
                deslocamento = 0;
                pacote = PRIMEIRO;
                return( OK );
        }
}

/*
 * Funcoes uteis
 */

/*
 * Inicializa buffer de sinalizacao
 */
limpasinal()
{
        respsinal->causa[0] = '\0';
        respsinal->diagnostico[0] = '\0';
        respsinal->ber[0] = '\0';
        respsinal->dur[0] = '\0';

        s_sinal = X25_NADA;
}

/*
 * limpa todos os buffers
 */
limpabuf()
{
        bufix25[0] = '\0';
        tambufix25 = 0;

        pacote = PRIMEIRO;
        deslocamento = 0;
}

/*
 * Copia o campo de causa
```

```
 */
cpcausa(apb)
char *apb;
{
        int i;

        for(i=0; *apb != SP && i < CMP_CAUSA; i++, apb++) {
                respsinal->causa[i] = *apb;
        }
        if(*apb != SP) {
                respsinal->causa[0] = '\0';
                return( CMP_CAUSA+1 );
        } else {
                respsinal->causa[i] = '\0';
                return(i);
        }
}

/*
 * Copia campo de diagnostico
 */
cpdiag(apb)
char *apb;
{
        int n;

        if((n = strlen(apb)) > CMP_DIAG) {
                respsinal->causa[0] = '\0';
                return(n);
        } else {
                strcpy(respsinal->diagnostico, apb);
                return(n);
        }
}
```

```
/*
 * Le as mensagens do cx25
 */
#include        "ix25.h"

extern struct saida fs [] [10];
extern int fpe [] [10];
extern int epcanal;
#define CR      'r'
#define LF      'n'

lemsg()
{
        char bufmsg[TAMMAXPAC+1], *apbufmsg, anterior, c;
        int i;

        anterior = NULL;
        i = 0;
        apbufmsg = bufmsg;
        while(scanf("%c", &c) == 1) {
                if(i++ > TAMMAXPAC + 2) {
                        fprintf(stderr, "mensagem erradal\n");
                        apbufmsg = bufmsg;
                        anterior = NULL;
                        i = 0;
                        return;
/*******
                        continue;
********/
                } else if(c != CR && c != LF && anterior != CR) {
                        *apbufmsg++ = c;
                        continue;
                }else if(c == CR && anterior != CR) {
                        anterior = CR;
                        continue;
                }else if(anterior == CR && c == LF) {
                        *apbufmsg = '\0';
                        apbufmsg = bufmsg;
                        anterior = NULL;
                        i = 0;
                        if(idmsg(bufmsg) < 0) {
                                fprintf(stderr, "mensagem errada2\n");
                        }
                        return;
/************
                        continue;
************/
                }else {
                        fprintf(stderr, "mensagem errada3 \n");
                        apbufmsg = bufmsg;
                        anterior = NULL;
                        i = 0;
                        return;
/***********
```

```
                        continue;
**************/
                }
        }
}
/*
 * Identifica e trata as mensagens, retorna MSGERR em caso de erro
 */
static char *tabmsg[] = {
        "*",
        ">",
        "\\",
        "\\COM",
        "\\CLR CONF",
        "\\NIVEL 3 OCUPADO",
        "\\NIVEL 3 DESOCUPADO",
        "\\ERR 1",
        "\\ERR 2",
        "\\ERR 3"
};

idmsg(apbuf)
char *apbuf;
{
        extern int tambuftrans;
        extern int buftrans[];
        int i, n, ret;

        if((*apbuf >= '@' && *apbuf <= 'O') || *apbuf == '\0') {
                ret = (*fs[MDADOS][epcanal].func)
                                (fs[MDADOS][epcanal].pl, apbuf);
                return( ret );
        }
        if(strcmp(tabmsg[0], apbuf) == 0) {
                ret = (*fs[PPTDAD][epcanal].func)
                                (fs[PPTDAD][epcanal].pl);
                if(tambuftrans == 0) {
                        epcanal = fpe[PPTDAD][epcanal];
                } else {
                        tambuftrans = 0;
                        buftrans[0] = '\0';
                }
                return( ret );
        }
        for(i=1, n=PPTCMD; n <= ERR3; i++, n++) {
                if(strcmp(tabmsg[i], apbuf) == 0) {
                        ret = (*fs[n][epcanal].func)
                                        (fs[n][epcanal].pl);
                        epcanal = fpe[n][epcanal];
                        return( ret );
                }
        }
        switch (*(apbuf+1)) {
        case 'A':
```

```
                ret = (*fs[ATENCAO][epcanal].func)
                                (fs[ATENCAO][epcanal].pl, apbuf);
                if(ret == OK) {
                        epcanal = fpe[ATENCAO][epcanal];
                }
                return( ret );
                break;
        case 'C':
                ret = (*fs[MCLR][epcanal].func)
                                (fs[MCLR][epcanal].pl, apbuf);
                if(ret == OK) {
                        epcanal = fpe[MCLR][epcanal];
                }
                return( ret );
                break;
        case 'R':
                ret = (*fs[RESET][epcanal].func)
                                (fs[RESET][epcanal].pl, apbuf);
                if(ret == OK) {
                        epcanal = fpe[RESET][epcanal];
                }
                return( ret );
                break;
        default:
                return( MSGERR );
                break;
        }
}
```

```
/*
 * Programa de testes da IX25
 */

#include        "ix25.h"
#include        "ix25global.h"

main(argc, argv)
int argc;
char *argv[];
{
        FILE *fp;
        int nent;       /* numero da entrada */

        if(argc != 2) {
                fprintf(stderr, "parametro errado\n");
                exit(1);
        }
        if((fp = fopen(*++argv, "r")) == NULL) {
                fprintf(stderr, "nao pode abrir %s\n", *argv);
                exit(1);
        }
        inic_cmd();     /* inicia cmd_x25 */
        printf("--------------------------------\n");
        printf("        ESTADO INICIAL\n");
        printf("--------------------------------\n");
        imp_saida();
        while(fscanf(fp, "%2d", &nent) == 1) {
                inic_cmd();
                switch(nent) {
                case 0:
                        ch_estabcv();
                        imp_saida();
                        break;
                case 1:
                        ch_limpacv();
                        imp_saida();
                        break;
                case 2:
                        ch_resetcv();
                        imp_saida();
                        break;
                case 3:
                        ch_trapacote();
                        imp_saida();
                        break;
                case 4:
                        ch_rcpacote();
                        imp_saida();
                        break;
                case 5:
                        ch_rcsinal();
                        imp_saida();
                        break;
```

```
case 6:
        ms_pptdad();
        imp_saida();
        break;
case 7:
        ms_pptcmd();
        imp_saida();
        break;
case 8:
        ms_fe();
        imp_saida();
        break;
case 9:
        ms_com();
        imp_saida();
        break;
case 10:
        ms_clrconf();
        imp_saida();
        break;
case 11:
        ms_ocupado();
        imp_saida();
        break;
case 12:
        ms_desocupado();
        imp_saida();
        break;
case 13:
        ms_err1();
        imp_saida();
        break;
case 14:
        ms_err2();
        imp_saida();
        break;
case 15:
        ms_err3();
        imp_saida();
        break;
case 16:
        ms_atencao();
        imp_saida();
        break;
case 17:
        ms_clr();
        imp_saida();
        break;
case 18:
        ms_reset();
        imp_saida();
        break;
case 19:
        ms_dados();
```

```
                        imp_saida();
                        break;
                default:
                        fprintf(stderr, "transicao invalida\n");
                        exit(1);
                        break;
                }
        }
        fclose(fp);
}

/*
 * inicializa cmd_x25 com EOF
 */
inic_cmd()
{
        FILE *fp2;
        char eof = EOF;

        if((fp2 = fopen("cmd_x25", "w")) == NULL) {
                fprintf(stderr,"nao pode abrir cmd_x25\n");
                return;
        }
        fprintf(fp2,"%c", eof);
        fclose(fp2);
}

/*
 * Chama a funcao ESTABCV
 */

ch_estabcv()
{
        static char *pcanal;    /* para retornar canal logico */
        int netd;       /* end. etd remoto */
        int codret;

        printf("--------------------------------\n");
        printf("          FUNCAO ESTABCV\n");
        printf("--------------------------------\n");
        fprintf(stderr, "\nEntre endereco ETD remoto (3 dig)\n");
        scanf("%3d", &netd);
        *pcanal = -1;

        codret = estabcv(netd, pcanal);

        printf("          SAIDA\n");
        printf("Codigo de Retorno = %d\n", codret);
        printf("Numero do Canal Logico Retornado = %d\n", *pcanal);
}

/*
 * Chama a funcao LIMPACV
 */
```

```
ch_limpacv()
{
        int canal, codret;

        printf("------------------------------\n");
        printf("        FUNCAO LIMPACV\n");
        printf("------------------------------\n");
        fprintf(stderr, "\nEntre com numero do canal logico\n");
        scanf("%d", &canal);

        codret = limpacv(canal);

        printf("        SAIDA\n");
        printf("Codigo de Retorno = %d\n", codret);
}

/*
 * Chama a funcao RESETCV
 */

ch_resetcv()
{
        int canal, codret;

        printf("--------------------------------\n");
        printf("        FUNCAO RESETCV\n");
        printf("--------------------------------\n");
        fprintf(stderr, "\nEntre com numero do canal logico\n");
        scanf("%d", &canal);

        codret = resetcv(canal);

        printf("        SAIDA\n");
        printf("Codigo de Retorno = %d\n", codret);
}

/*
 * Chama a funcao TRAPACOTE
 */

ch_trapacote()
{
        int canal, codret, tamanho;
        char pbuffer[TAMMAXPAC+1];

        printf("----------------------------------\n");
        printf("        FUNCAO TRAPACOTE\n");
        printf("----------------------------------\n");
        fprintf(stderr, "\nEntre com numero do canal logico\n");
        scanf("%d", &canal);
        fprintf(stderr, "Entre com os dados\n");
        scanf("%s", pbuffer);
        fprintf(stderr, "Entre com tamanho dos dados\n");
        scanf("%d", &tamanho);
```

```
        codret = trapacote(canal, pbuffer, tamanho);

        printf("        SAIDA\n");
        printf("Codigo de Retorno = %d\n", codret);
}

/*
 * Chama a funcao RCPACOTE
 */

ch_rcpacote()
{
        int codret;
        char bufet[TAMMAXPAC + 1];
        struct rcpacres *respet;

        printf("----------------------------\n");
        printf("        FUNCAO RCPACOTE\n");
        printf("----------------------------\n");

        codret = rcpacote(bufet, respet);

        printf("        SAIDA\n");
        printf("Codigo de Retorno = %d\n", codret);
        printf("        BUFFERS ET\n");
        printf("Buffer de dados da ET = %s\n", bufet);
        printf("Numero de bytes transferidos = %d\n", respet->nbtransf);
        printf("Canal logico = %d\n", respet->canal);
        printf("        VARIAVEIS DA IX25\n");
}

/*
 * Chama a funcao RCSINAL
 */

ch_rcsinal()
{
        int sinal;
        struct rcsinres *rcsinet;
        struct rcsinres etrcsinres;

        rcsinet = &etrcsinres;

        printf("----------------------------\n");
        printf("        FUNCAO RCSINAL\n");
        printf("----------------------------\n");

        sinal = rcsinal(rcsinet);

        printf("        SAIDA\n");
        printf("Sinalizacao = %d\n", sinal);
        printf("        BUFFER DE SINALIZACAO DA ET\n");
        printf("Canal logico = %d\n", rcsinet->s_canal);
        printf("Causa = %s\n", rcsinet->causa);
```

```
        printf("Diagnostico = %s\n", rcsinet->diagnostico);
        printf("Endereco ETD remoto = %s\n", rcsinet->ber);
        printf("        VARIAVEIS DA IX25\n");
}

/*
 * Recebe mensagem PPTDAD
 */

ms_pptdad()
{
        printf("-------------------------\n");
        printf("        MSG PPTDAD\n");
        printf("-------------------------\n");
        fprintf(stderr, "\nEntre com *rn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem PPTCMD
 */

ms_pptcmd()
{

        printf("-------------------------\n");
        printf("        MSG PPTCMD\n");
        printf("-------------------------\n");
        fprintf(stderr, "\nEntre com >rn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem FE
 */

ms_fe()
{
        printf("-------------------------\n");
        printf("        MSG FE\n");
        printf("-------------------------\n");
        fprintf(stderr, "\nEntre com \\rn\n");
        getchar();

        lemsg();
```

```
        printf("        SAIDA\n");
}

/*
 * Recebe mensagem COM
 */

ms_com()
{
        printf("------------------------\n");
        printf("        MSG COM\n");
        printf("------------------------\n");
        fprintf(stderr, "\nEntre com \\COMrn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem CLRCONF
 */

ms_clrconf()
{
        printf("------------------------\n");
        printf("        MSG CLR CONF\n");
        printf("------------------------\n");
        fprintf(stderr, "\nEntre com \\CLR CONFrn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem OCUPADO
 */

ms_ocupado()
{
        printf("------------------------\n");
        printf("        MSG OCUPADO\n");
        printf("------------------------\n");
        fprintf(stderr, "\nEntre com \\NIVEL 3 OCUPADOrn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}
```

```
/*
 * Recebe mensagem DESOCUPADO
 */

ms_desocupado()
{
        printf("------------------------\n");
        printf("        MSG DESOCUPADO\n");
        printf("------------------------\n");
        fprintf(stderr, "\nEntre com \\NIVEL 3 DESOCUPADOrn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}
/*
 * Recebe mensagem ERR1
 */

ms_err1()
{
        printf("------------------------\n");
        printf("        MSG ERR1\n");
        printf("------------------------\n");
        fprintf(stderr, "\nEntre com \\ERR 1rn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem ERR 2
 */

ms_err2()
{
        printf("------------------------\n");
        printf("        MSG ERR2\n");
        printf("------------------------\n");
        fprintf(stderr, "\nEntre com \\ERR 2rn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem ERR3
 */
```

```
ms_err3()
{
        printf("--------------------------\n");
        printf("        MSG ERR3\n");
        printf("--------------------------\n");
        fprintf(stderr, "\nEntre com \\ERR 3rn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem ATENCAO
 */

ms_atencao()
{
        printf("---------------------------\n");
        printf("        MSG ATENCAO\n");
        printf("---------------------------\n");
        fprintf(stderr, "\nEntre com \\ATENCAO xxxrn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem CLR
 */

ms_clr()
{
        printf("----------------------------\n");
        printf("        MSG CLR\n");
        printf("----------------------------\n");
        fprintf(stderr, "\nEntre com \\CLR xxx yyyrn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe mensagem RESET
 */

ms_reset()
{
        printf("-----------------------\n");
```

```
        printf("        MSG RESET\n");
        printf("-----------------------\n");
        fprintf(stderr, "\nEntre com \\RESET xxx yyyrn\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Recebe DADOS
 */

ms_dados()
{
        printf("-----------------------\n");
        printf("        MSG DADOS\n");
        printf("-----------------------\n");
        fprintf(stderr, "\nEntre com os dados (20)\n");
        getchar();

        lemsg();

        printf("        SAIDA\n");
}

/*
 * Imprime as Variaveis Externas
 */

imp_saida()
{
        FILE *fpl;
        char cmds[TAMMAXPAC + 5];

        printf("Estado Atual da IX25 = %d\n", epcanal);

        if((fpl = fopen("cmd_x25", "r")) == NULL) {
                fprintf(stderr, "nao pode abrir cmd_x25\n");
        } else {
                *cmds = '\0';
                fscanf(fpl, "%s", cmds);
                fclose(fpl);
                printf("Comando Enviado = %s\n", cmds);
        }
        printf("        SINALIZACAO\n");
        printf("Sinalizacao = %d\n", s_sinal);
        printf("causa = %s\n", respsinal->causa);
        printf("diagnostico = %s\n", respsinal->diagnostico);
        printf("ber = %s\n", respsinal->ber);
        printf("        BUFFERS INTERNOS\n");
        printf("Buffer de Dados = %s\n", bufix25);
        printf("Tamanho do Buffer de Dados = %d\n", tambufix25);
```