I.C.M.S.C.

INSTITUTO DE CIÊNCIAS MATEMÁTICAS DE SÃO CARLOS

NÚCLEO BÁSICO PARA UM SISTEMA DISTRIBUÍDO DE TEMPO REAL

CELIO ESTEVAN MORON

UNIVERSIDADE DE SÃO PAULO

SÃO CARLOS - SÃO PAULO
BRASIL

NÚCLEO BÁSICO PARA UM SISTEMA DISTRIBUÍDO DE TEMPO REAL

CELIO ESTEVAN MORON

Dissertação de Mestrado apresentada na Área "Ciências da Computação" do Instituto de Ciências Matemáticas de São Carlos da Universidade de São Paulo.

Orientador: Prof. Dr. ARTHUR JOÃO CATTO

- São Carlos -

1986

Dedico esta obra à

Roxana

e aos meus pais.

A G R A D E C I M E N T O S

Ao professor Dr. Arthur João Catto pela orientação.

A todas as pessoas do Departamento de Computação e Estatística da UFSCar que de alguma forma colaboraram neste trabalho.

NÚCLEO BÁSICO PARA UM SISTEMA DISTRIBUÍDO DE TEMPO REAL

R E S U M O

Com a evolução da tecnologia passamos a ter disponíveis processadores de baixo custo, o que tornou possível a construção de Sistemas Distribuídos que anteriormente eram economicamente impraticáveis. Por causa desse desenvolvimento o conhecimento de técnicas de Programação Concorrente e de Projeto de Sistemas Distribuídos tem se tornado importante para muitas aplicações.

Este trabalho descreve a implementação de um Núcleo Básico para um Sistema Distribuído de Tempo Real que, associado a um hardware adequado, forma uma máquina distribuída básica, capaz de receber o código objeto de vários processos a serem executados simultaneamente. Para satisfazer às condições de tempo real, cada processador será dedicado a um único processo. Não há qualquer compartilhamento de memória e a única maneira de dois processos se comunicarem é através de uma chamada remota de procedimento.

Por causa dos requisitos de confiabilidade do sistema, são previstos mecanismos que, passado um tempo máximo estipulado sem que se obtenha uma resposta, tomam as providências necessárias para que não ocorram travamentos de processos("deadlocks").

BASIC KERNEL FOR A REAL TIME DISTRIBUTED SYSTEM

A B S T R A C T

With the evolution of technology, inexpensive processors have become available which allow the construction of distributed systems that were previously economically impracticable. Because of such development, the knowledge of concurrent programming and distributed systems design techniques have become important to many applications.

This dissertation describes the implementation of a "Basic Kernel for a Real-Time Distributed System" which, together with an appropriate hardware, forms a basic distributed machine, able to receive object code of several processes to be simultaneously executed. To satisfy real-time constraints each processor will be dedicated to a single process. There is no shared memory, and the only way for communication between processes is a Remote Procedure Call.

Because of system reliability requirements, mechanisms are provided which, after waiting for a predetermined time without obtaining a reply, take the necessary steps to prevent process deadlock.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

O modelo de arquitetura tradicional de von Neumann, que chamaremos *sistema*, é composto de memória, processador e portas de entrada e saída, através das quais o sistema pode se comunicar com dispositivos periféricos.

Um *programa seqüencial* é um conjunto de operações que são executadas pelo processador, uma a uma, como passos da resolução de um problema. Estas operações são totalmente ordenadas no tempo e o resultado de uma operação pode ser usado por outra. A atividade gerada pela execução de um programa sequencial é chamada *processo*.

Na memória são armazenados o programa que está sendo executado, os dados que estão sendo processados e também os resultados finais e intermediários produzidos. O processador realiza os cálculos especificados pelo programa em execução que podem envolver operações aritméticas, lógicas ou de controle. As portas de entrada e saída fornecem o meio pelo qual os dados

e programas são transmitidos do meio externo para o sistema, e os resultados obtidos do sistema para o meio externo. Aumentando a velocidade do processador teremos as operações sendo executadas mais rapidamente, melhorando conseqüentemente o desempenho do sistema. No entanto, por mais que aumentemos a velocidade do processador, não conseguiremos fazer o desempenho do sistema passar de um certo limite, pois esbarraremos na limitação imposta pela velocidade dos dispositivos periféricos que são muito lentos se comparados com o processador. O processador executará rapidamente as operações internas e ficará ocioso esperando as operações de entrada e saída se completarem.

Uma outra maneira de melhorar o desempenho de um sistema é através da execução de um programa concorrente. Um *programa concorrente* especifica dois ou mais sub-programas seqüenciais que podem ser executados simultaneamente como processos paralelos. Este aumento de desempenho do sistema deve-se ao fato de o tempo ocioso do processador, enquanto uma operação de entrada ou saída é executada, poder ser usado para execução de outro processo, e à possibilidade de compartilhamento de recursos entre vários processos.

Quando um programa concorrente é executado partilhando um processador pela comutação dos processos no processador, temos um *sistema multiprogramado* |1|. Neste caso o processador, após disparar uma operação de entrada ou saída ou executar um processo durante algum tempo, comuta o processo salvando as informações relevantes, possibilitando assim a continuação da execução deste processo posteriormente.

Um programa concorrente pode também ser executado por um sistema composto por dois ou mais processadores comunicantes. Se os processadores partilham uma memória comum temos um *sistema multiprocessador* |2|, e se forem interligados por um sistema de comunicação temos um *sistema distribuído*.

O modelo de arquitetura tradicional de von Neumann obriga os problemas a serem resolvidos através de programas seqüenciais. Desta forma o paralelismo natural existente num modelo a ser programado, em vez de facilitar a solução, através da execução de mais de um processo ao mesmo tempo, torna-se uma dificuldade adicional.

Com a evolução da tecnologia passamos a ter disponíveis processadores de baixo custo, o que tornou possível a construção de sistemas multiprocessadores e sistemas distribuídos que anteriormente eram economicamente inviáveis. Por causa deste desenvolvimento o conhecimento de técnicas de programação concorrente tem se tornado importante para muitas aplicações como por exemplo sistemas operacionais, bases de dados distribuídas, computação científica com alto grau de paralelismo, programação em tempo real, sistemas de controle, etc.

O *estado* de um sistema pode ser definido pelo conteúdo da memória e dos registradores, a cada instante durante a execução de um programa. Assim, o conteúdo dessas áreas de armazenamento, no início da execução de um programa, é definido como o estado inicial do sistema. Cada passo da execução do programa transforma o estado existente em um novo estado, através da modificação de uma ou mais dessas áreas de

armazenamento. Num dado instante, o estado de cada um dos processadores componentes de um sistema distribuído é chamado *estado local*, e o conjunto de todos os estados locais é chamado *estado global*.

A primeira conseqüência da existência de um sistema de comunicação em um sistema distribuído é a imposição de um atraso na troca de informações entre os processadores. Desta forma o conhecimento do estado global de um sistema distribuído torna-se muito difícil, uma vez que, ao término da leitura dos estados locais, esta já poderá estar desatualizada, devido aos atrasos decorrentes das comunicações. Nos sistemas distribuídos temos que trabalhar levando em conta apenas os estados locais, já que a manutenção do estado global é inviável. Os sistemas cujo estado global pode ser conhecido são chamados *sistemas centralizados*, e como exemplos temos os sistemas baseados no modelo de von Neumann, sistemas multiprogramados e sistemas multiprocessadores.

Num sistema distribuído as informações tornam-se distantes por não poderem ser conhecidas diretamente, dando uma dimensão de *espaço* ao sistema, e também por não poderem ser movidas instantâneamente de um processador para outro, dando uma dimensão de *tempo* ao sistema.

A execução de um programa concorrente dá origem a um conjunto de processos paralelos. Em um sistema centralizado tais processos coexistem e a comunicação entre eles é feita via compartilhamento de memória, de forma sempre instantânea. Em um sistema distribuído os processos podem ser *remotos*, isto é, podem estar em processadores diferentes e se comunicarem pela

troca de mensagens através do sistema de comunicação. Isto faz com que os sistemas distribuídos sejam potencialmente pouco sensíveis a falhas que possam ocorrer em algum processador, predispondo esses sistemas a serem muito confiáveis.

O atraso observado na comunicação entre processos depende da arquitetura do sistema de comunicação. A mudança do sistema de comunicação pode ter como resultado a mudança das características do sistema distribuído. Neste trabalho, supomos que o sistema de comunicação não perca mensagens e que o atraso da mensagem seja variável, finito e que não comprometa o desempenho do sistema distribuído.

Uma falha detectada num sistema distribuído pode ser isolada através da passagem de todas as funções realizadas pelo processador defeituoso para um outro processador que esteja disponível. Isto é feito da seguinte forma: quando um processador não consegue comunicar-se com outro ou quando as informações que recebe não são consistentes, ele comunica o fato ao processador responsável pela reconfiguração do sistema distribuído, que por sua vez passa a enviar mensagens de teste a todos os processadores. Identificado o processador defeituoso, o sub-programa correspondente é transferido para outro processador que possa executar a mesma função. Quando o processador está ligado a equipamentos que não podem ser substituídos, ativa-se um circuito que comuta o processador defeituoso com um processador reserva.

1.1 COMUNICAÇÃO ENTRE PROCESSOS

Comunicação é o mecanismo usado para a troca de informações entre processos de acordo com as necessidades, podendo influir na execução desses processos.

Nos sistemas centralizados a comunicação entre processos é caracterizada pelos seguintes fatos: existe uma memória partilhada, processos coexistem em uma só máquina, e uma falha é catastrófica e paralisa todo o sistema. Nestes sistemas o atraso observado na comunicação entre processos deve-se quase que totalmente ao atraso no acesso à memória partilhada.

Em um sistema distribuído os processos não coexistem em uma só máquina, e podem ser remotos. Isto quer dizer que dois processos podem ser executados em máquinas diferentes e, ainda assim, cooperar na realização de uma tarefa. Nesses sistemas temos a necessidade da troca explícita de mensagens devido à inexistência de memória partilhada.

Processos remotos cooperam em um sistema distribuído por diversas razões, como por exemplo:

1- Podemos ter programas usuários independentes executando computações não relacionadas, mas que por razões de eficiência, cooperam partilhando recursos. Exemplo: programas executados ao mesmo tempo em um sistema distribuído que dispõe de uma única impressora.

2- Podemos ter programas usuários independentes executando computações relacionadas. Neste caso a execução de um programa pode influenciar a de outro e até mesmo alterar seus resultados. Exemplo: programas executados em um sistema

distribuído que têm acesso a uma mesma base de dados.

3- Podemos ter um único programa usuário dando origem a vários processos relacionados que cooperam de acordo com determinações do usuário. Exemplo: um programa executado em um sistema distribuído que faça o controle de tráfego de uma ferrovia.

1.2 MECANISMOS DE COMUNICAÇÃO E SINCRONIZAÇÃO

A comunicação entre processos concorrentes permite que um processo influencie a execução de outro. Quando dois processos se comunicam é preciso que haja sincronização para que a ordem de ocorrência de eventos seja estabelecida. A comunicação entre processos pode ser feita por meio de variáveis partilhadas ou da troca de mensagens.

A comunicação entre processos por meio de variáveis partilhadas adapta-se a sistemas com partilhamento de memória como sistemas multiprogramados e multiprocessadores. Por outro lado, em um sistema distribuído a comunicação entre processos tem que ser por troca de mensagens, uma vez que o sistema não dispõe de memória partilhada.

O modo como esta troca de mensagens é feita divide os sistemas distribuídos em duas categorias: *sistemas orientados por mensagem* e *sistemas orientados por procedimento*.

A comunicação em um sistema orientado por mensagem é uma extensão da comunicação feita por meio de variáveis partilhadas para um ambiente distribuído onde, em vez da memória, usa-se um sistema de comunicação. Existem dois comandos básicos que

operam sobre estas mensagens: um para enviar uma mensagem para outro processador e um para receber uma mensagem de outro processador. O comando receptor espera até que uma mensagem esteja disponível e então, depois de recebê-la, permite a continuação do processo. Da mesma forma, o comando emissor retém o controle até que a mensagem seja enviada.

A comunicação em um sistema orientado por procedimento é uma extensão da chamada de procedimentos em linguagens seqüenciais para um sistema distribuído. As mensagens trocadas pelos processos consistem de chamadas de procedimentos externos ou dos resultados da execução desses procedimentos externos. O processo que chama o procedimento externo é denominado *cliente* e o que executa o procedimento, *servidor*. Em geral, o processo cliente fica bloqueado até que os resultados da execução do procedimento externo sejam recebidos. Desta forma o controle é passado do processo cliente para o processo servidor.

Os sistemas orientados por mensagem e orientados por procedimento são duais, isto é, todo sistema orientado por mensagem tem o seu correspondente nos sistemas orientados por procedimento e vice-versa |19|.

O desenvolvimento de mecanismos para sincronização de processos começou com o uso de semáforos e regiões críticas proposto por Dijkstra |3|,|4|. Seguiu-se uma série de propostas evoluindo esta idéia |5|,|6|,|7|,|8| que aumentaram sua eficiência, confiabilidade e generalidade. Depois de semáforos e regiões críticas tivemos o desenvolvimento de outros mecanismos para comunicação entre processos e gerenciamento de recursos partilhados, tais como regiões críticas condicionais

|9| e monitores |10|,|11|.

Estes mecanismos foram incorporados a linguagens de programação concorrente de alto nível, sendo duas das mais importantes: Pascal Concorrente |12| e Modula 2 |13|.

Modula 2 foi projetada com vistas à multiprogramação, enquanto Pascal Concorrente requer para sua implementação um único processador ou um multiprocessador com memória partilhada. Desta forma estas linguagens não são ideais para uma rede de microprocessadores com memória distribuída.

O próximo passo foi o projeto de linguagens de programação concorrentes para ambientes com memória distribuída. Nesta direção temos Communicating Sequential Processes (CSP) |14| e Distributed Processes (DP) |15| que eliminam a comunicação entre processos por meio de variáveis partilhadas e introduzem a comunicação por meio da troca de mensagens.

1.3 ORGANIZAÇÃO DO TRABALHO

No capítulo 2 é feita uma caracterização de Sistemas Distribuídos para melhor entendermos estes sistemas e prever possíveis conseqüências do seu uso.

O capítulo 3 analisa a comunicação e sincronização entre processos em um Sistema Distribuído bem como a linguagem DP e suas características de tempo real.

Um Núcleo Básico para um Sistema Distribuído de Tempo Real é descrito no capítulo 4. Neste Núclo Básico não existe qualquer compartilhamento de memória e os processos se comunicam por meio de Chamada Remota de Procedimento.

O capítulo 5 descreve um Sistema de Comunicação que serve de suporte ao núcleo Básico fazendo a troca de mensagens.

No apêndice A1 estão contidas as principais listagens do "Nucleo Básico para um Sistema Distribuído de Tempo Real" e no apêndice A2 as do Sistema de Comunicação.

CAPÍTULO 2

CARACTERIZAÇÃO DE UM SISTEMA DISTRIBUÍDO

2.1. DEFINIÇÕES

Um Sistema de Computação oferece um conjunto de funções que são utilizadas através de um Sistema Operacional. Estas funções são executadas por processos chamados *entidades lógicas*.

Segundo Le Lann |16|, um Sistema de Computação é composto por entidades lógicas e_j, que são agrupadas formando conjuntos E_i com a finalidade de cooperarem na realização de uma função f_i.

$F = \{ f_i, i \in I \}$ é o conjunto das funções realizadas pelo sistema.

$E_i = \{ e_j^i, j \in J(i) \}$ é o conjunto de entidades que participam da função f_i, onde $J(i)$ é o conjunto das entidades que podem participar da funcão f_i.

$s_t(e_j^i)$ é o estado instantâneo da entidade lógica e_j^i no

instante t.

O vetor $S_t(E_i) = \{ s_t(e_j^i),\ j \in J(i) \}$ é o estado global do conjunto de entidades lógicas E_i no instante t.

Um sistema é *δ_i-centralizado* se existe $k \in J(i)$ tal que e_k^i conhece $S_t(E_i)$.

Um sistema é *totalmente centralizado* se é *δ_i-centralizado* para todo $i \in I$.

Um sistema é *δ_i-distribuído* se não existe $k \in J(i)$ tal que e_k^i conhece $S_t(E_i)$.

Um sistema é *totalmente distribuído* se é *δ_i-distribuído* para todo $i \in I$.

2.2. SISTEMA DISTRIBUÍDO GENÉRICO

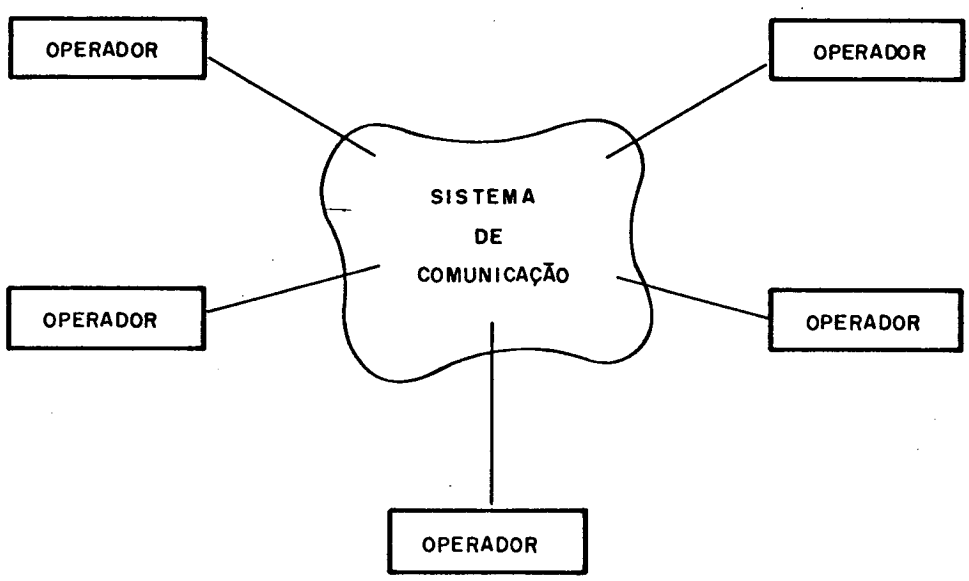


Figura 2.1 Sistema Distribuído

Um Sistema Distribuído consiste de um número finito de módulos autônomos de processamento chamados processadores operadores. Esses operadores são interligados por um Sistema de Comunicação que permite a troca de informações entre eles (Figura 2.1).

Os processadores operadores são compostos por um processador e memória primária, e opcionalmente possuem dispositivos periféricos (Figura 2.2).

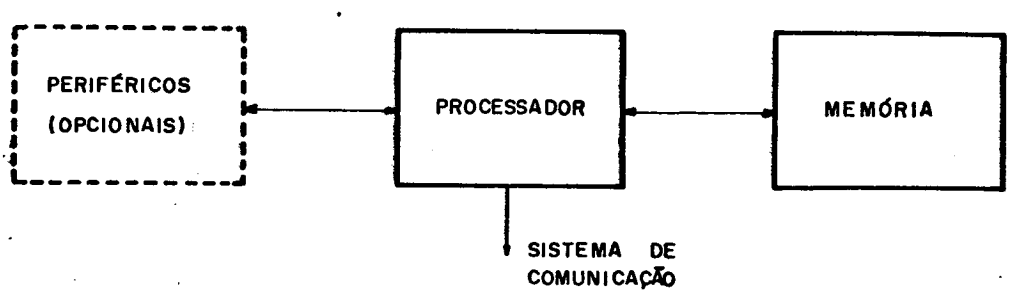


Figura 2.2 Processador Operador

A única maneira de dois operadores se comunicarem é através da troca de mensagens, que se realiza de acordo com um protocolo de comunicação.

Protocolo de comunicação é um conjunto de regras que definem como é feita a troca de mensagens entre operadores. Para reduzir a complexidade do projeto, o protocolo é dividido em muitos níveis, sendo cada um destes níveis construído sobre o seu predecessor (Figura 2.3).

O nível mais elementar é constituído pela ligação física entre os processadores operadores que compõem o Sistema

Distribuído. Os níveis sucessivos usam os serviços oferecidos pelos níveis imediatamente inferiores para fornecer um serviço mais sofisticado.

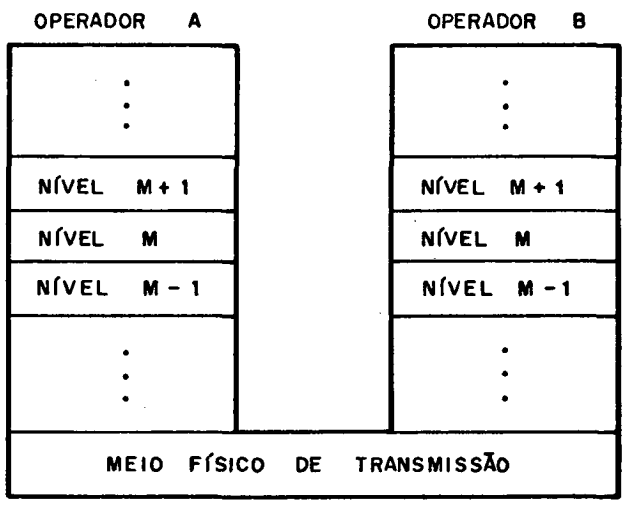


Figura 2.3 Organizaçao do protocolo em níveis

Para se fazer a comunicaçao entre dois níveis adjacentes é necessária a definição de uma interface entre eles. Esta interface define os serviços disponíveis, como ter acesso a eles e quais os formatos e convenções usadas.

2.3. *PROTOCOLOS*

A ISO (International Standard Organization) definiu um modelo de referência, visando a padronização de protocolos |17|. O modelo ISO define níveis de protocolos de comunicação com funções específicas isoladas para cada nível. No modelo ISO o sistema de comunicação inclui somente os três primeiros níveis do protocolo (Figura 2.4).

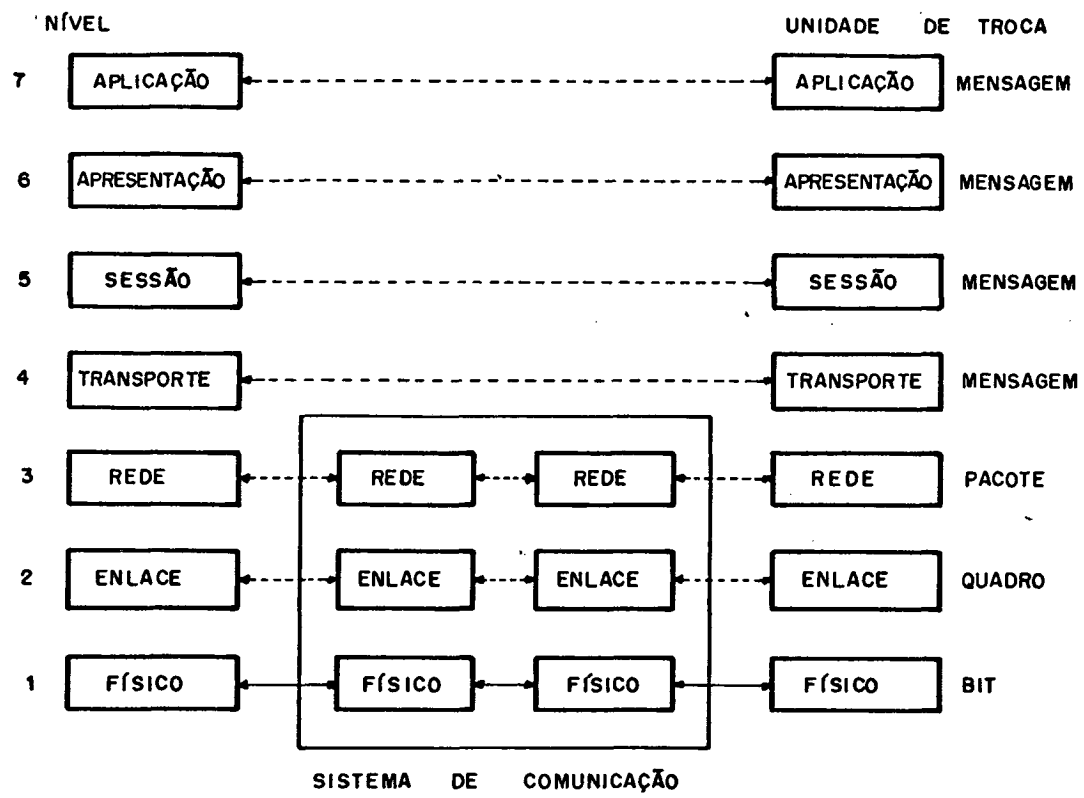


Figura 2.4 Arquitetura de Interconexão da ISO

1- Nível Físico

O nível físico refere-se à transmissão e recepção de cadeias de bits em um canal de comunicação sem considerar o conteúdo da estrutura.

2- Nível de Enlace

A tarefa do nível de enlace é transmitir e receber uma cadeia fazendo com que a mesma chegue ao receptor livre de erros. Isto é feito pela quebra dos dados de entrada em quadros de dados, transmissão dos quadros seqüencialmente e processamento dos quadros de reconhecimento. Este nível cria e

reconhece o limite do quadro através da colocação de bits especiais no começo e fim do quadro.

Se um quadro for perdido ou recebido com erros ele deve ser retransmitido. Múltiplas transmissões de um mesmo quadro podem permitir a duplicação de quadros. Por exemplo, se um quadro de reconhecimento for perdido, um quadro já recebido será enviado novamente.

Este nível resolve os problemas de erros, perda ou duplicação de quadros, de modo que o nível 3 possa admitir estar trabalhando sobre uma linha livre de erros.

3- Nível de Rede

Este nível determina as características da interface entre o Sistema de Comunicaçao e o Operador. Suas funções são: aceitar mensagens do processador operador, convertê-las em pacotes e encaminhar os pacotes ao seu destino. O nível de rede assegura que todos os pacotes sejam recebidos corretamente no seu destino e na ordem própria.

Os pacotes são encaminhados pelo Sistema de Comunicação com base em algoritmos de rota que tentam minimizar o atraso imposto por este sistema. O caminho é obtido através de uma tabela de rotas que é mantida pelo algoritmo.

Os algoritmos de direcionamento podem ser *determinísticos* ou *adaptativos*. Os determinísticos são baseados no endereço do processador operador destino e topologia da rede. Os adaptativos são baseados no endereço do processador operador destino e no estado corrente da rede.

4- Nível de Transporte

Este nível é conhecido como nível *operador-operador*. A função deste nível é aceitar dados do seu nível superior, dividí-los em unidades menores, se necessário, e passá-los ao nível de rede.

O nível 4 é um nível *fonte-destino*, isto é, um processo em uma máquina fonte pode conversar com um processo numa máquina destino. Os níveis inferiores são analisados pelo sistema de comunicação, enquanto este nível e superiores são analisados pelos processadores operadores.

5- Nível de Sessão

Neste nível são estabelecidos o início e o encerramento do fluxo de dados de estação a estação. Ele permite a dois usuários estabelecerem uma conexão que envolve a troca de parâmetros, tais como autenticação do usuário, modo de transmissão, etc.

6- Nível de Apresentação

É o nível de apresentação que manipula a conversão dos dados de um formato para outro e envolve compactação, descompactação, mudança do conjunto de caracteres e criptografia.

7- Nível de Aplicação

O conteúdo do nível de aplicação é determinado pelo usuário. Este nível serve diretamente ao usuário, provendo serviços apropriados para o gerenciamento do sistema. Este gerenciamento cumpre funções como iniciar, manter e terminar

conexões para transferência de dados entre processos de aplicacão. Os outros níveis existem somente para suportar este nível.

2.4. SISTEMA DE COMUNICAÇÃO

Um Sistema de Comunicação |18| é um meio de transmissão de mensagens entre processadores operadores. Devido a restrições físicas o Sistema de Comunicação não constitui um meio perfeito de comunicação. Devemos então considerar a existência de atrasos na propagação de mensagens por um Sistema de Comunicação.

O atraso de propagação é o tempo necessário para uma mensagem ser transmitida entre dois processadores operadores. Esse atraso é variável, finito e seu valor não é conhecido com precisão. O atraso de propagação é variável porque poderá ocorrer erro de transmissão ou perda de mensagem, o que implicará em nova transmissão e, conseqüentemente, um tempo maior para que a mensagem seja recebida no destino.

Um Sistema de Comunicação introduz as noções de *espaço* e *tempo*. Espaço porque a informação torna-se distante quando não está no mesmo processador operador e tempo porque a informação não pode ser movida instantâneamente.

Existem dois tipos de Sistemas de Comunicação: ponto-a-ponto e por difusão (broadcast). Todo Sistema de Comunicação consiste de dois componentes básicos: *linhas de transmissão* e *interfaces processadoras de mensagens (IPM)*.

O Sistema de Comunicação ponto-a-ponto envia uma mensagem de uma IPM para outras IPMs intermediárias, sucessivamente até que a mensagem chegue à IPM destino. Em cada IPM intermediária a mensagem é armazenada até que possa ser transmitida para outra IPM. Este Sistema de Comunicação também é chamado store-and-forward.

Num Sistema de Comunicação por difusão existe um canal de comunicação partilhado por todas as IPMs. Uma mensagem é enviada por alguma IPM e recebida por todas as outras. Se uma mensagem não for endereçada a uma IPM, esta a ignora.

Num Sistema de Comunicação do tipo ponto-a-ponto a IPM é denominada Processador de Comunicação, enquanto num Sistema de Comunicação por difusão a IPM é chamada Interface de Comunicação.

Na figura 2.5 temos um exemplo de Sistema de Comunicação ponto-a-ponto.

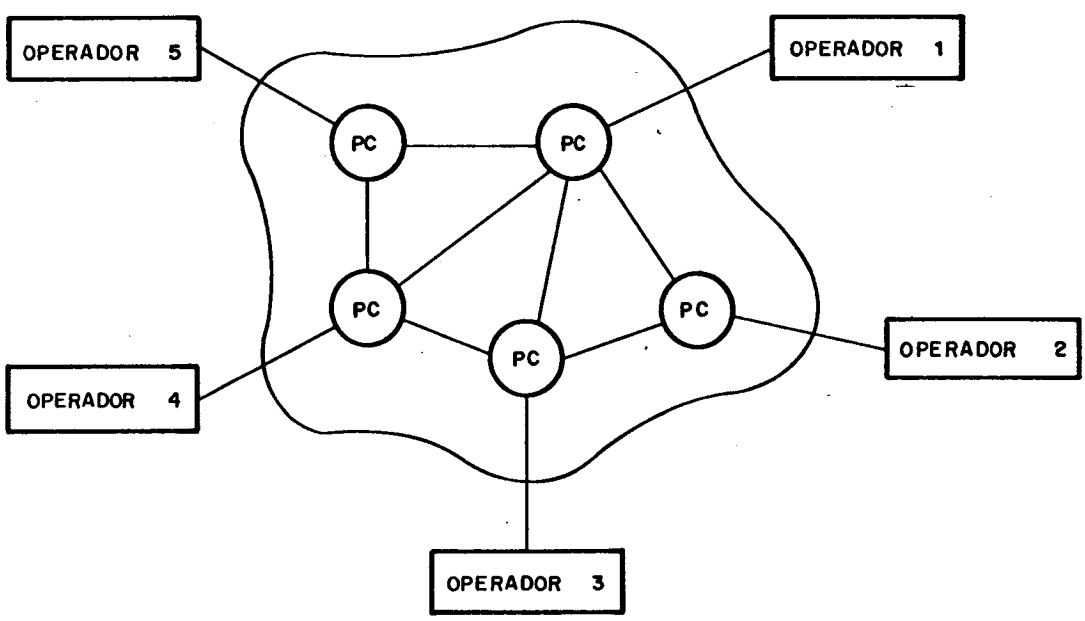


Figura 2.5 Sistema de Comunicação ponto-a-ponto

Na figura 2.6 temos um exemplo de Sistema de Comunicação por difusão.

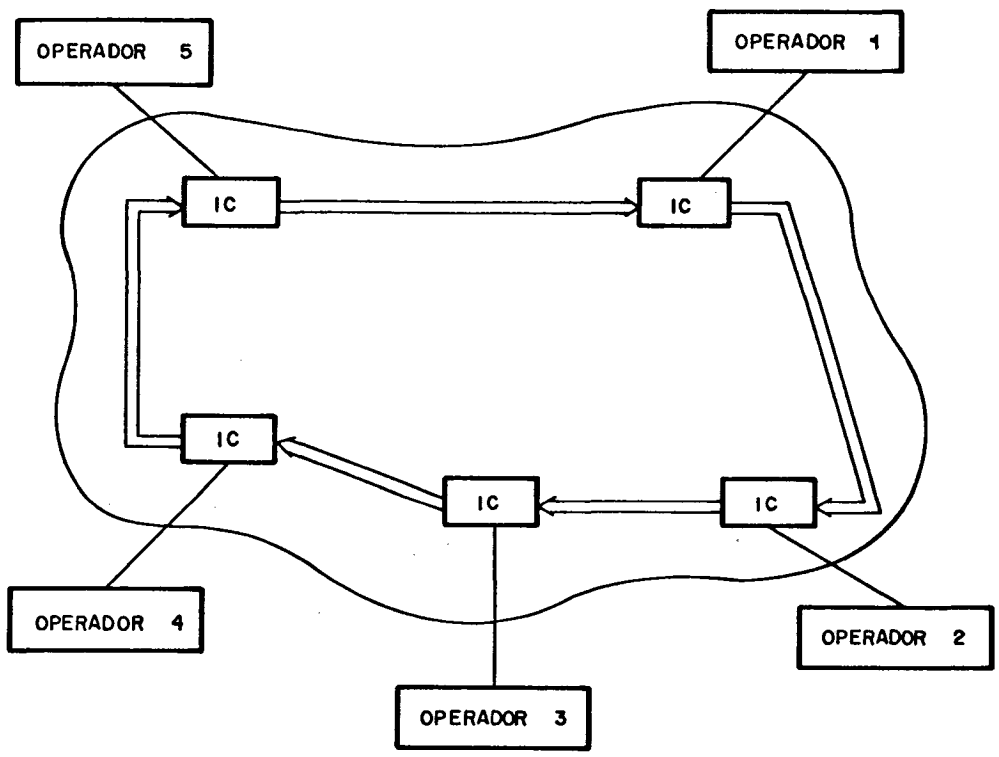


Figura 2.6 Sistema de Comunicação por difusão

2.5. *ESPAÇO E TEMPO*

Sistemas distribuídos são sistemas onde processadores operadores cooperam entre si, mas sem partilhar o mesmo espaço físico e não tendo, portanto, a mesma referência de tempo. Como vimos, todo Sistema de Comunicação tem um atraso de propagação da mensagem maior que zero, o que impossibilita a definição de uma referência de tempo comum a todos os operadores.

Suponhamos a existência de uma referência absoluta como uma referência ideal para melhor entendermos o comportamento do Sistema.

A Figura 2.7 mostra três eventos que se originam no processador operador B e são relatados para dois outros processadores operadores, onde o sistema é visto por uma referência absoluta.

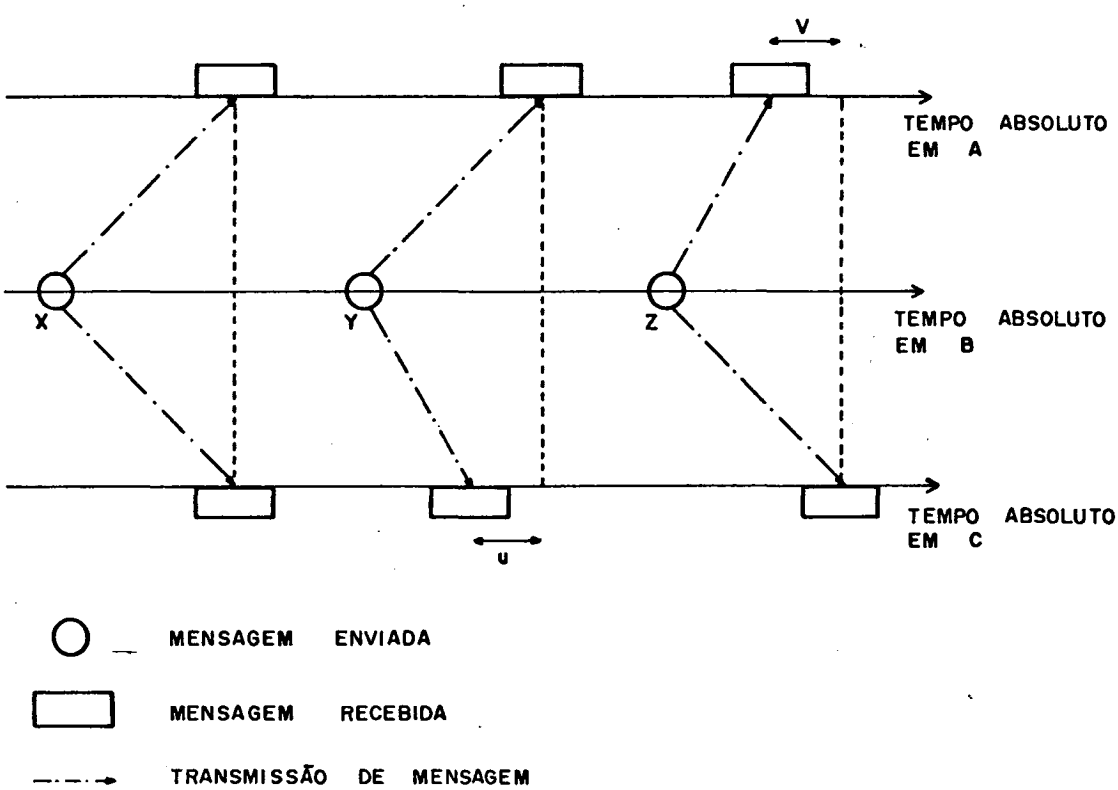


Figura 2.7 Transmissão de três eventos

Podem ocorrer três casos distintos, e o modo com que os operadores vêem um determinado evento pode não coincidir. Podemos ter o caso onde o evento X é visto ao mesmo tempo pelos

processadores operadores A e C. Temos outro caso onde o evento Y é visto por C e, após decorrido um intervalo de tempo u, é visto por A. E, finalmente, o caso onde o evento Z é visto por A e, após decorrido um intervalo de tempo v, é visto por C.

Um processador operador pode ter uma visão parcial e coerente do sistema distribuído ou uma visão completa mas incoerente do sistema distribuído. Neste contexto, visão coerente significa que os eventos são vistos na mesma ordem em que ocorreram na referência absoluta.

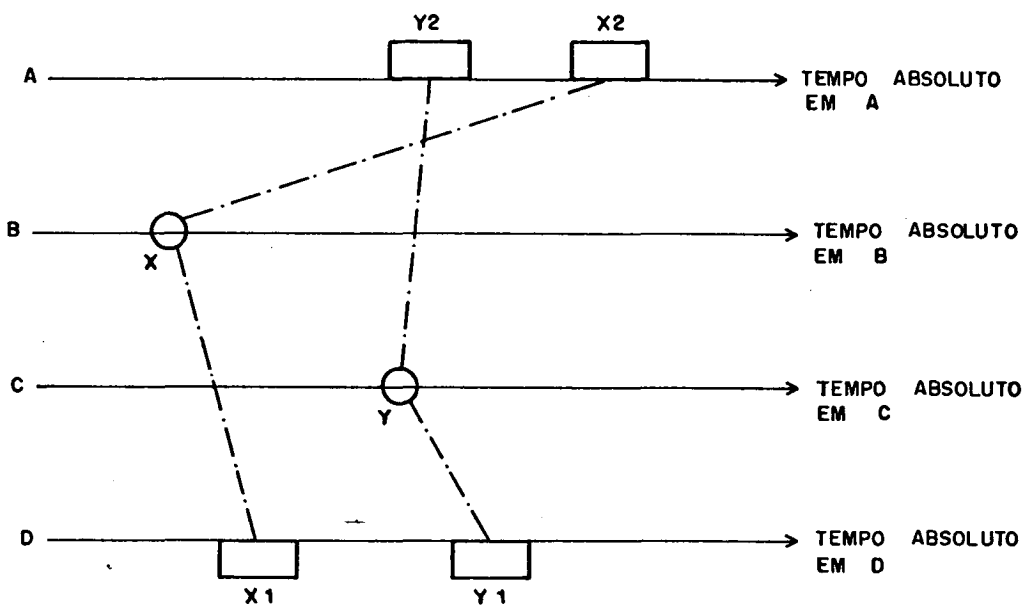


Figura 2.8 Visão de dois eventos

A figura 2.8 mostra um evento X que é transmitido para dois processadores operadores, e um evento Y que ocorre depois de X e é transmitido para os mesmos processadores operadores. O processador operador A vê o evento Y acontecendo antes que X, enquanto o operador D vê os eventos na mesma ordem em que

aparecem na referência absoluta.

2.6. PRINCÍPIOS PARA UM SISTEMA DISTRIBUÍDO

Le Lann|16| enumera três princípios básicos para sistemas distribuídos que são:

1- Princípio da não determinação do tempo

Para uma dada seqüência de eventos, é impossível provar que dois processadores operadores diferentes podem observar esta seqüência da mesma forma. Um exemplo que ilustra este princípio está na figura 2.8.

Como o estado interno de um processador operador é passado através de um sistema de comunicação temos o seguinte princípio:

2- Princípio da observação relativa

Em um sistema distribuído, é impossível provar que dois processadores operadores quaisquer têm a mesma visão global de um sub-conjunto do sistema.

Conseqüentemente, um processador operador não conhece o par (t, e(t)) de outro processador operador com absoluta certeza, onde t é o tempo e e(t) o estado local de um processador operador. Isto pode ser contornado supondo que o estado local e(t) é válido durante um intervalo de tempo pré-definido t com alguma probabilidade de acerto. Então temos o seguinte princípio:

3- Princípio da não determinação do estado

O estado instantâneo de um operador pode ser expresso somente em termos de valores possíveis, associados a alguma probabilidade. Isto significa que o estado global de um sistema distribuído não existe. Como conseqüência, de acordo com a definição da seção 1 deste capítulo o sistema distribuído genérico que estamos analisando é totalmente distribuído.

Os problemas a serem resolvidos através de sistemas distribuídos devem levar em conta que o controle pode ser realizado sem o conhecimento do estado global, cuja construção é inviável. Desta forma cada processador operador trabalhará com uma aproximação deste estado e nenhum deles isoladamente será encarregado de realizar todas as funções de controle, elas serão realizadas pelo conjunto de processadores operadores.

2.7. *TERMINOLOGIA*

Um sistema distribuído pode ter o software fracamente acoplado ou fortemente acoplado. Se o software for fracamente acoplado a cooperação entre os processadores operadores é limitada a alguns serviços padrões simples, tais como transferência de arquivos, acesso remoto, etc. Se o software for fortemente acoplado a cooperação é mais elaborada e envolve o partilhamento para múltiplos recursos remotos.

Um sistema fracamente acoplado em geral é chamado Rede de Computadores, enquanto um fortemente acoplado constitui um sistema distribuído propriamente dito. Todas as vezes que nos referirmos neste texto a um Sistema Distribuído estaremos considerando um Sistema Distribuído fortemente acoplado.

CAPÍTULO 3

COMUNICAÇÃO ENTRE PROCESSOS EM UM SISTEMA DISTRIBUÍDO

3.1. COMUNICAÇÃO E SINCRONIZAÇÃO DE PROCESSOS

Processos concorrentes podem se comunicar e sincronizar de acordo com a necessidade. A comunicação entre processos concorrentes permite que um processo influencie a execução de outro. Muitas vezes quando dois processos se comunicam é preciso que haja sincronização para que a ordem de ocorrência de eventos seja estabelecida.

A comunicação é feita através de uma ação que é detectada pelo outro processo. Esta ação pode ser a alteração do valor de alguma variável ou o envio de alguma mensagem. No primeiro caso dizemos que a comunicação é feita por meio de *variáveis partilhadas* e no segundo caso por meio da *troca de mensagens*.

A comunicação entre processos por meio de variáveis partilhadas adapta-se a sistemas de computação com memória comum, como sistemas multiprogramados e multiprocessadores.

Regiões críticas, monitores e semáforos são exemplos de mecanismos de comunicação de processos baseados no uso de variáveis partilhadas.

Por outro lado, em um sistema distribuído a comunicação entre processos tem que ser por troca de mensagens, uma vez que o sistema não dispõe de memória partilhada.

Os sistemas distribuídos podem ser de duas categorias dependendo de como implementam e usam a noção de processo e sincronização. Estas duas categorias são denominadas sistemas orientados por mensagem e sistemas orientados por procedimento |19|.

A comunicação em um sistema orientado por mensagem é uma extensão daquela feita por meio de variáveis partilhadas para um ambiente distribuído, onde em vez da comunicação ocorrer por meio da memória, ela é feita através de um sitema de comunicação. Este tipo de sistema tem a vantagem de permitir implementação simples e eficiente, e introduzir diretamente o paralelismo |20|. Por outro lado, este tipo de sistema tem diversas desvantagens do ponto de vista do projeto da linguagem. Temos a introdução de uma primitiva cujo controle é muito diferente do mecanismo de controle dos procedimentos. Isto não é desejável em linguagens baseadas em procedimentos, como Algol ou Pascal, onde a troca de mensagens requer uma nova primitiva e não simplesmente uma extensão da chamada local de procedimento. Outra desvantagem é a introdução de inconsistências dentro da linguagem, na estrutura de verificação da compatibilidade de tipos.

A comunicação em um sistema orientado por procedimento é uma extensão da chamada de procedimento para um sistema distribuído. Sua maior vantagem é a padronização das chamadas locais de procedimentos e chamadas externas de procedimentos. O programador pode não perceber que vários processos do seu programa são executados em diferentes processadores operadores, ou seja, isto pode ficar transparente ao programador. Desta forma o programador não tem que sair do seu ambiente usual de programação para se preocupar com detalhes de troca de mensagens, e tudo se passa como se estivesse trabalhando com um sistema centralizado. Por outro lado o maior problema deste tipo de sistema está na propriedade que os procedimentos têm de sempre retornarem ao ponto da chamada. Garantir que uma chamada externa de procedimento sempre retorne adequadamente é extremamente difícil em um sistema distribuído, onde um processador ou o sistema de comunicação podem falhar. Por causa do exposto tem sido difícil a extensão de chamadas locais para remotas, sem a introdução de algum mecanismo deselegante para controle do tempo de resposta, códigos de erros ou armadilhas para detecção de falhas.

3.2. SISTEMAS ORIENTADOS POR MENSAGEM

Esta classe de sistemas possui primitivas para enviar e receber mensagens. Uma *mensagem* é uma estrutura de dados usada para enviar informações de um processador operador para outro. Um *nome* é um identificador pelo qual uma particular mensagem pode ser reconhecida.

Uma mensagem pode ser enviada de um processo para outro pela execução da seguinte primitiva:

ENVIA nome

PARA destino

A mensagem contém o valor da estrutura de dados *nome* no momento em que a primitiva *ENVIA* é executada. O *destino* indica para qual processo esta mensagem deve ser enviada.

Uma mensagem pode ser recebida por um processo pela execução da seguinte primitiva:

RECEBE nome

DE origem

Esta primitiva atribui à estrutura de dados *nome* o valor da mensagem recebida. A *origem* dá ao programador controle sobre os pontos de onde a mensagem pode provir.

Chamaremos o processo que executa a primitiva *ENVIA* de *processo emissor*, e o processo que executa o *RECEBE* correspondente de *processo receptor*.

3.2.1. SINCRONIZAÇÃO

Uma primitiva de sincronização *ENVIA* pode causar um atraso no seu processo emissor até que o *RECEBE* correspondente seja executado pelo processo receptor. Uma primitiva é dita *não bloqueada* se sua execução nunca atrasa o processo emissor e *bloqueada* caso possa atrasar o seu processo emissor |21|.

Numa primitiva não bloqueada, as mensagens são colocadas em um buffer durante o intervalo entre o envio e o recebimento. Se o buffer estiver cheio quando um *ENVIA* é executado, há duas opções: o *ENVIA* pode atrasar o processo emissor até que exista lugar no buffer para a mensagem, ou o *ENVIA* pode retornar um código para o processo emissor, indicando que, o buffer esta cheio e a mensagem não pode ser enviada. Da mesma forma, um *RECEBE* pode atrasar o processo receptor até que uma mensagem esteja disponível ou então retornar um código para o processo receptor, indicando que nenhuma mensagem está disponível.

A troca de mensagens usando buffer, permite que o processo emissor envie uma mensagem antes que o *RECEBE* correspondente seja executado. Este modo de troca de mensagens é também chamado *assíncrono*.

A troca de mensagens sem usar buffer faz com que a execução de um *ENVIA* atrase o processo emissor até que o *RECEBE* correspondente seja executado. Só então a mensagem é transferida e processada. Este modo de troca de mensagens é também chamado *síncrono*. Na troca síncrona de mensagens a transferência da mensagem é o ponto de sincronização entre o processo emissor e o receptor.

Um RECEBE bloqueado implementa implicitamente a sincronização entre o emissor e o receptor, uma vez que o RECEBE espera até que a mensagem seja enviada. Um comando de troca de mensagem bloqueado pode ter o mesmo efeito semântico que o não bloqueado correspondente, pelo uso da *comunicação seletiva* que é baseada no comando com guarda de Dijkstra |23|.

Em um comando de comunicação seletiva, comandos com guarda têm a seguinte forma:

guarda --> Lista de comandos

O guarda consiste de uma expressão booleana, opcionalmente seguida por um comando de troca de mensagem. O guarda obtém êxito se a expressão booleana for verdadeira e a execução do comando de troca de mensagem não causar atraso; o guarda falha se a expressão booleana for falsa; o guarda nem falha nem obtém êxito se a expressão booleana for verdadeira mas o comando de troca de mensagem não puder ser executado sem causar atraso. Quando uma lista de comandos com guarda é selecionada para execução, seus comandos são executados sucessivamente da esquerda para a direita.

Comandos com guarda permitem a construção de comandos alternativos e repetitivos com comportamento não-determinístico.

O comando alternativo tem a seguinte forma:

```
IF  G1 --> S1
/   G2 --> S2
    ...
/   Gn --> Sn
end
```

Quando um comando alternativo é alcançado, todos os seus guardas são avaliados simultaneamente. Se ao menos um guarda obtém êxito, um deles é selecionado não deterministicamente; o comando de troca de mensagem correspondente é executado (se presente) e a lista de comandos que segue o guarda é executada.

Se todos os guardas falharem o comando aborta. Se todos os guardas não falham nem obtêm êxito, a execução é atrasada até que algum guarda tenha êxito.

O comando repetitivo tem a seguinte forma:

```
DO  G1 --> S1
/   G2 --> S2
    ...
/   Gn --> Sn
end
```

A execução do comando repetitivo é semelhante à do comando alternativo, sendo que a seleção e execução de um comando com guarda é repetida até que todos os guardas falhem, com o tempo o comando repetitivo termina em vez de abortar.

3.3. SISTEMAS ORIENTADOS POR PROCEDIMENTO

Em um sistema orientado por procedimento temos uma primitiva de comunicação e sincronização de processos denominada *chamada remota de procedimento*. Os processos cooperam entre si, enviando e recebendo mensagens através do Sistema de Comunicação, sendo que estas mensagens são chamadas para procedimentos externos, e resultados da execução de procedimentos externos. Durante a execução de uma chamada remota de procedimento, o processo que solicita a operação é denominado *cliente*, enquanto que o processo que a executa é denominado *servidor*. O nome da primitiva deriva do fato de um processo cliente "chamar" um procedimento que é executado em uma máquina remota, por um processo servidor.

O processo cliente e o processo servidor executam duas trocas de mensagem, da seguinte maneira: o processo cliente *ENVIA* a chamada e, em seguida, *RECEBE* os resultados, enquanto o processo servidor *RECEBE* a chamada e, depois, *ENVIA* os resultados.

Do ponto de vista do processo cliente, a chamada remota de procedimento é executada do seguinte modo: a identificação do procedimento e os valores dos argumentos de entrada são enviados para o servidor apropriado e o processo cliente é atrasado, até que os resultados tenham sido recebidos e atribuídos aos argumentos de saída correspondentes.

O processo servidor, por sua vez, recebe uma chamada para um procedimento, executa os comandos correspondentes e envia os resultados para o processo cliente.

Uma chamada remota de procedimento só termina depois que o cliente receber o resultado desejado. Uma chamada remota de procedimento é uma extensão natural de uma chamada de procedimento em uma linguagem seqüencial. Numa linguagem seqüencial, nós temos a transferência do controle para sub-programas em um mesmo processador operador, enquanto na chamada remota de procedimento temos a transferência do controle para sub-programas em processadores operadores diferentes.

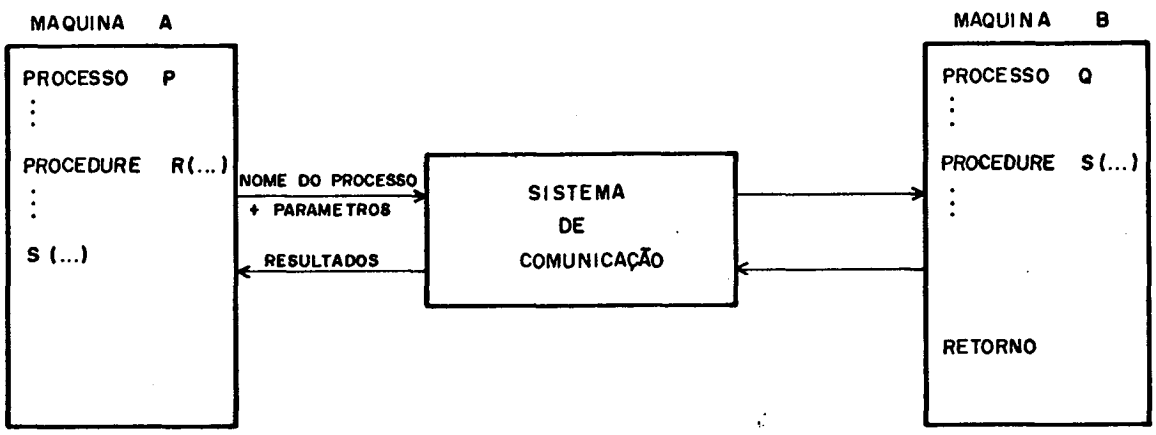


Figura 3.1 Chamada remota de procedimento

Temos na figura 3.1 uma chamada remota de procedimento do cliente A para o servidor B. Na máquina A temos o procedimento R, no processo P, que faz uma chamada remota para o procedimento S que está na máquina B, processo Q. A máquina B executa o procedimento S com os parâmetros fornecidos pelo cliente e transmite os resultados de volta para ele.

3.4. LINGUAGENS PARA PROCESSAMENTO DISTRIBUÍDO

Communicating Sequential Processes (CSP) de Hoare |14| é um exemplo de linguagem de programação orientada por mensagem. A proposta de Hoare é uma linguagem que pode ser implementada em uma máquina convencional ou numa rede de processadores.

Distributed Processes (DP) de Brinch Hansen |15|, é um exemplo de linguagem de programação orientada por procedimento. Brinch Hansen propõe uma linguagem de programação concorrente para uma rede de microcomputadores com memória distribuída, voltada para aplicações em tempo real. Os processos comunicam-se e sincronizam-se por meio de chamadas remotas de procedimento e regiões com guardas.

Lauer e Needham |19| demonstram que os sistemas orientados por mensagem e os sistemas orientados por procedimento são duais e o sistema que é construído de acordo com um modelo tem um correspondente direto no outro modelo. Sistemas duais são logicamente idênticos. O desempenho de um sistema para um modelo é idêntico ao do seu sistema dual, isto quer dizer que as primitivas para um modelo podem ser tão eficientes quanto as do seu dual.

Como CSP e DP representam abordagens duais escolhemos uma delas para fazermos uma análise mais aprofundada. A escolha recaiu sobre DP devido à padronização das chamadas locais de procedimentos e chamadas remotas de procedimentos. O tratamento que é dado aos parâmetros da chamada remota também é o mesmo das chamadas locais. Com isto temos programas muito semelhantes aos programas para sistemas centralizados.

3.5 *CARACTERÍSTICAS DE UM PROGRAMA EM TEMPO REAL*

Programas convencionais descrevem ações que são executadas por algum processador em um tempo finito. Este tempo não é apenas uma característica do programa mas também do processador empregado. Por causa das vantagens da generalização da

construção de programas, estes são geralmente projetados de tal forma que seus resultados sejam independentes da velocidade de execução dos seus processadores. Este tipo de programa é chamado *independente do tempo*. A única asserção indispensável para este tipo de programa é que a velocidade do processador seja maior que zero. A maior vantagem dos programas não dependentes do tempo é que a prova de sua validade pode ser deduzida somente do texto do programa.

Existem certos programas que interagem com o meio externo de forma não determinística. Esses programas podem esperar por sinais que indiquem o fim de uma tarefa executada cooperativamente, ou sinais que indiquem a ocorrência de algum evento externo. Esses programas dependem necessariamente da velocidade do processador. Um programa que dependa da velocidade do processador é chamado de programa em *tempo real* ou *programa dependente do tempo* |22|. Um exemplo de programa dependente do tempo, é um programa que recebe sinais de um sensor, que os emite a cada vez que é sensibilizado pelo meio externo, e que, sem ter o reconhecimento de que um sinal foi recebido, emite um outro sinal. A execução correta deste programa vai depender do menor intervalo de tempo entre dois sinais consecutivos do sensor.

Um programa em tempo real interage com o meio externo, no qual muitas coisas podem acontecer simultaneamente, a alta velocidade, recebendo uma variedade de requisições não determinísticas. O programa não pode predizer a ordem em que estas requisições serão feitas, mas deve poder responde-las dentro de um certo limite de tempo. Caso contrário, uma entrada de dados pode ser perdida ou os resultados podem perder seu

significado. Tipicamente, um programa em tempo real controla um sistema com uma configuração fixa de processos e periféricos, e realiza um número fixo de tarefas concorrentes. Esse programa em tempo real nunca termina, continuando a servir ao meio externo indefinidamente.

3.6. A LINGUAGEM DP

Um programa em DP consiste de um número fixo de processos seqüenciais que começam simultaneamente e existem para sempre. Para satisfazer as condições de tempo real cada processador será dedicado a um único processo.

Um processo define suas variáveis locais, alguns procedimentos globais e um comando inicial, da seguinte maneira:

PROCESS nome

Variáveis locais

Procedimentos globais

Comando inicial

Um processo pode ter acesso somente a suas variáveis locais; não existem variáveis globais. No entanto, um processo pode chamar procedimentos globais definidos dentro dele próprio ou de outro processo. Uma chamada remota de procedimento de um processo para outro constitui uma *requisição externa*, e é a única forma de comunicação entre processos.

Um processo executa dois tipos de operações: o comando inicial e requisições externas. Essas operações alternam-se no tempo, por intercalação. Um processo começa executando o seu comando inicial, e continua até que este termine ou tenha que esperar que uma condição torne-se verdadeira. Então, o processo inicia uma nova operação, em conseqüência de uma chamada externa. Quando esta operação terminar ou tiver que esperar alguma condição, o processo iniciará outra operação ou retomará uma que tenha sido suspensa à espera de uma condição que tenha-se tornado verdadeira. Esta intercalação entre o comando inicial e as requisições externas continua indefinidamente. Mesmo que o comando inicial termine, o processo continuará a existir e poderá ainda aceitar requisições externas. A substituição da operação sendo executada acontece somente quando ela termina ou espera por uma condição.

Um processo continua a executar operações até que todas as suas operações correntes tenham sido suspensas, ou até que ele faça uma requisição para um outro processo. No primeiro caso, o processo fica ocioso até receber uma nova requisição externa. No segundo caso, o processo fica ocioso até que o outro processo complete a operação requisitada. Fora isto, não se faz qualquer suposição sobre a ordem em que um processo realiza suas operações.

Um procedimento define seus parâmetros de entrada e saída, variáveis locais e um comando que é executado quando ele é chamado.

PROCEDURE nome (parâm. de entrada × parâm. de saída)

Variáveis locais

Comando

Um processo P pode chamar um procedimento R definido dentro de um outro processo Q como segue:

CALL Q.R (expressões, variáveis)

Antes que a operação R seja executada, os valores das expressões da chamada são atribuídos aos parâmetros de entrada. Quando a operação termina, os valores dos parâmetros de saída são atribuídos às variáveis da chamada.

O espaço de variáveis locais de um processo é criado somente no início da execução. O espaço de variáveis locais de um procedimento é criado cada vez que esse procedimento é ativado. Dessa forma, os procedimentos declarados num mesmo processo podem-se comunicar através das variáveis locais desse processo. Como, em qualquer instante, existe no máximo uma operação ativa num processo, o acesso a essas variáveis é seguro e livre de conflitos.

O não determinismo pode ser controlado por dois tipos de comandos chamados comandos com guardas e regiões com guardas, cuja operação é semelhante àquela descrita em 3.2.

C A P Í T U L O 4

NÚCLEO BÁSICO PARA UM SISTEMA DISTRIBUÍDO DE TEMPO REAL

4.1. INTRODUÇÃO

Neste capítulo discute-se o projeto e construção de um *NÚCLEO BÁSICO PARA UM SISTEMA DISTRIBUÍDO*, que possibilite a implementação de uma linguagem para processamento distribuído orientada por procedimento, tomando como base Distributed Processes (DP) |15| de Brinch Hansen. Cada processador operador do sistema distribuído terá uma cópia do Núcleo Básico.

Um programa distribuído consiste de um conjunto de processos seqüenciais, sendo que o código objeto correspondente a cada processo é carregado em um só processador. Todos os processos são ativados simultaneamente. O Núcleo Básico possibilita que o código objeto gerado para um dado processo seja armazenado no processador operador correspondente e executado posteriormente. Temos portanto o sistema distribuído mais o Núcleo Básico formando uma *máquina distribuída básica*

capaz de receber e executar o código objeto gerado por um compilador.

Cada processador hospeda e executa um único processo. Durante a execução de um processo, seu comando inicial e as chamadas externas recebidas por seus procedimentos são atendidos intercaladamente. O processador hospedeiro começa executando o comando inicial de seu processo até que este acabe ou seja suspenso à espera de que alguma condição torne-se verdadeira. Nesse momento é selecionada uma chamada externa de procedimento para ser executada. Essa ação também continua até que o procedimento acabe ou seja suspenso esperando por alguma condição, quando uma nova chamada externa ou uma ação suspensa anteriormente será selecionada para execução. Temos então uma alternância entre a execução do comando inicial e as diversas chamadas para os procedimentos deste processo. Mesmo após o comando inicial ter terminado, o processo continuará existindo e atendendo as chamadas para os seus procedimentos.

Supomos a existência de um Sistema de Comunicação que pode ser do tipo ponto-a-ponto ou difusão. No capítulo seguinte discutiremos o particular Sistema de Comunicação usado como suporte para este trabalho.

As chamadas remotas serão feitas através do Sistema de Comunicação e executarão concorrentemente com as outras chamadas remotas e o comando inicial. Se olharmos nosso sistema como um todo veremos um Sistema Distribuído, mas se olharmos um único processador operador veremos um Sistema Multiprogramado que executa o comando inicial e chamadas remotas para os procedimentos do único processo a ele associado. O processador

operador executará alternadamente o comando inicial e as chamadas remotas, dando impressão ao meio externo de que os processos são executados em paralelo. Chamaremos o comando inicial e as chamadas remotas de procedimento indistintamente de comando.

4.2. PARTILHAMENTO DO PROCESSADOR

Para que um processador tenha uma execução pseudo-paralela são necessários mecanismos capazes de interromper este processador durante a execução do código objeto de um comando e fazer com que ele passe a executar o código objeto de outro comando.

AMBIENTE VOLÁTIL |24| de um comando é definido como o conjunto de informações que têm que ser preservadas para que o comando possa ser restaurado depois de suspenso. Se isto não for feito o comando poderá ser retomado em um estado diferente daquele em que foi interrompido. As informações que têm que ser preservadas são as que correm o risco de serem perdidas caso um outro comando use o processador operador. O ambiente volátil inclui informações tais como o conteúdo dos registradores, registros gerenciadores da memória e o contador de programa.

Os diversos comandos são executados no processador operador trocando o ambiente volátil atual pelo novo ambiente volátil. Desta forma toda vez que um comando for suspenso haverá uma troca de ambiente volátil, transparente para os comandos envolvidos, que não percebem que não têm o uso exclusivo do processador operador.

Desta forma temos que definir uma estrutura de dados para armazenar o ambiente volátil de um comando enquanto ele não usa o processador operador. Nós chamaremos esta estrutura de dados de Descritor de Comando. O conjunto dos descritores dos comandos de um processo é chamado Descritor de Processo. Temos então que um processo é caracterizado pelo seu código objeto e pelo descritor de processo correspondente (figura 4.1).

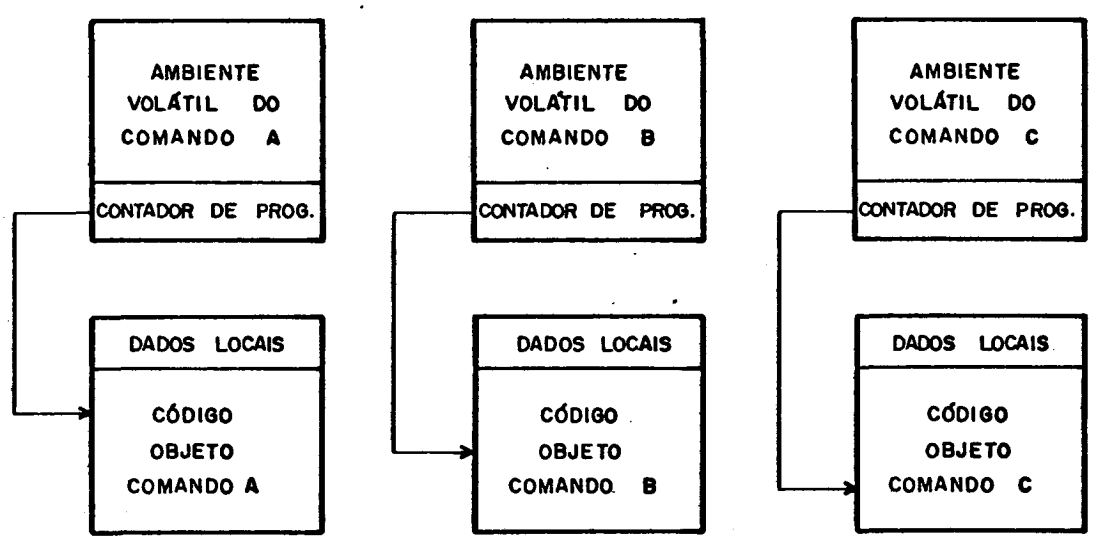


Figura 4.1 Constituição dos Comandos

O Comutador de Comandos é o mecanismo que quando ativado armazena no descritor do processo correspondente os valores que compõem o ambiente volátil do comando corrente e o substitui pelo conteúdo do ambiente volátil do comando que vai ser executado.

4.3. PARTILHAMENTO DA MEMÓRIA PRINCIPAL

Vimos anteriormente que o processador operador será partilhado por diversos comandos; temos então que prover memória para que estes comandos possam ser executados de maneira eficiente. Um mesmo comando terá várias execuções ao mesmo tempo devido a várias chamadas remotas para ele. A maneira mais fácil de abordar este problema é alocando um espaço de armazenamento diferente para cada execução de um mesmo comando. Este espaço de armazenamento conterá uma cópia do código objeto do comando além de sua área de dados locais (figura 4.2).

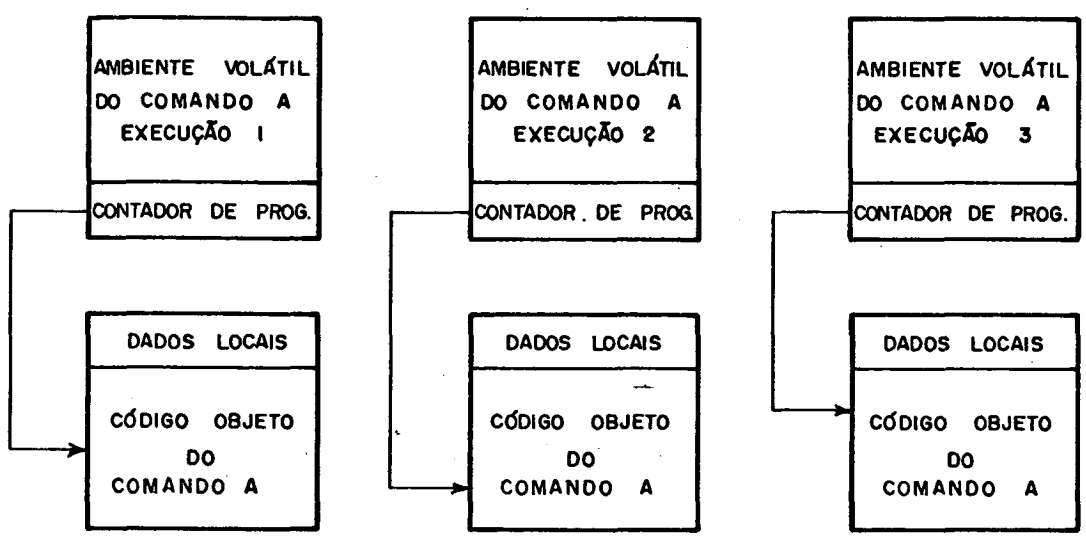


Figura 4.2 Diferentes execuções de um mesmo comando

Podemos economizar espaço de memória usando um único conjunto de código objeto para cada comando ao invés de uma cópia para cada chamada. Isto é, múltiplas chamadas remotas para um procedimento partilham o mesmo conjunto de código

objeto.

Um comando consiste de um conjunto de código objeto mais dados locais. Esta área de armazenamento local é utilizada pelas variáveis de trabalho temporário e poderá ser alterada quando houver a comutação da execução corrente para a execução de uma outra chamada remota, causando o funcionamento inadequado do nosso sistema. Este problema pode ser superado através da utilização de código puro.

4.3.1 COMANDO PURO OU REENTRANTE

Nós podemos separar os dados locais do conjunto de código objeto, para que este conjunto possa ser ativado mais que uma vez simultaneamente. Garantimos desta forma que uma execução não altera os dados locais de outra execução do mesmo comando. Comandos com esta propriedade são chamados puros ou reentrantes.

Um comando puro age sobre diferentes áreas de dados locais dependendo da ativação corrente. Ele pode ter acesso a estes dados de uma forma indireta, através de um ponteiro de relocação que é associado a cada execução do comando. Este ponteiro de relocação contém o endereço do começo dos dados locais de uma determinada execução (figura 4.3).

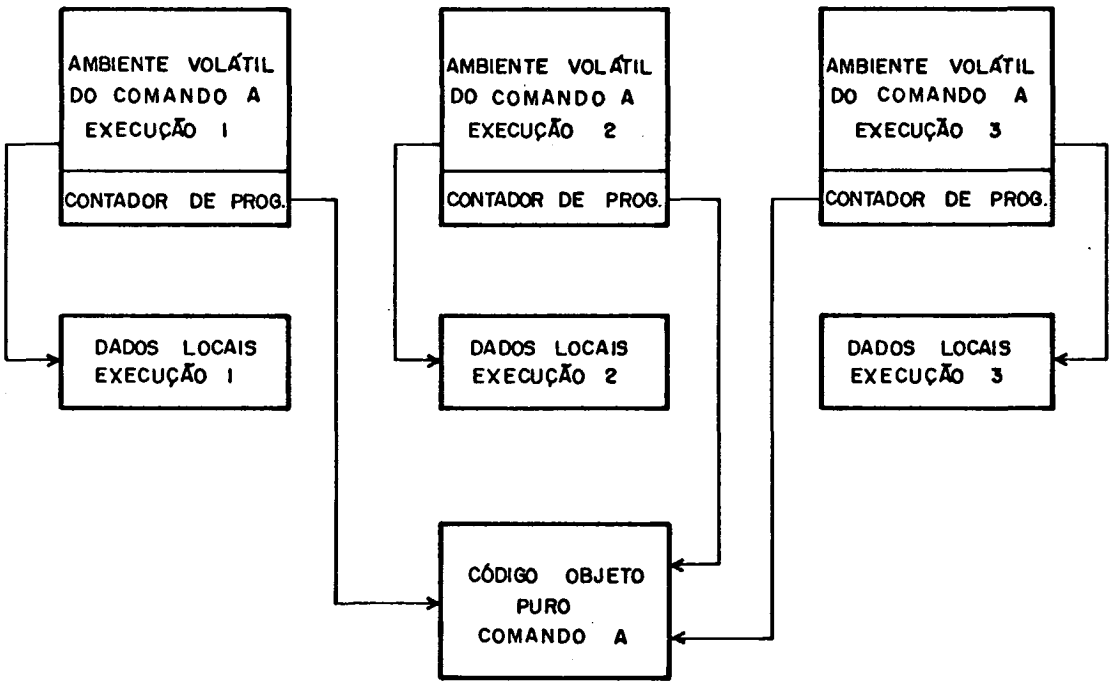


Figura 4.3 Comando puro ou reentrante

Na figura 4.3 temos o mesmo conjunto de código objeto de um comando sendo utilizado por três execuções diferentes uma vez que eles utilizam diferentes áreas de dados locais. Cada execução não tem consciência do fato de outras chamadas usarem o mesmo conjunto de código.

4.4. ESTRUTURA DO NÚCLEO

Um processo ao fazer uma chamada remota fica bloqueado esperando o seu resultado; portanto cada processo terá no máximo um pedido de execução pendente de cada um dos outros processos. Como no nosso sistema cada processador executa um único processo, uma tabela com um número de entradas igual ao número de processadores pode guardar todas as informações necessárias para que o comando inicial e todas as chamadas externas de procedimentos eventualmente suspensas sejam retomados futuramente (figura 4.4).

	ESTADO	REGISTROS	CONTADOR DE PROGR.	PONTEIRO DE RELOCAÇÃO
1				
2				
3				
4				
5				
6				
N				

Figura 4.4 Tabela descritora de processos

Cada entrada é o descritor de um comando. O índice indica o número do processo que fez a chamada remota e o índice igual ao número deste processo refere-se à execução do comando inicial. O campo *estado* é um valor booleano que, quando verdadeiro, indica que esta chamada remota está suspensa e, caso contrário, que não existe chamada remota deste processador. O campo *registradores* é o conjunto de registradores do processador operador. O *contador de programa* fornece o endereço onde o processo foi interrompido e o *ponteiro de relocação* fornece o endereço do início dos dados locais desta execução.

Estrutura de dados da tabela de processos:

```
CONST N = Número-de-Processos;
TYPE Descritor-de-Comando = REGISTRO
                                Estado: Booleano;
                                Registradores: ... ;
```

```
                    Contador-de-programa: ... ;
                    Ponteiro-de-Relocação: ... ;
                END;
VAR Tabela-Comandos:Arranjo |1..N| de Descritor-de-Comando;
    c-cor: inteiro; (* Comando corrente *)
    Tabela-Vazia: booleano;
```

Um processo pode fazer uma chamada remota de qualquer procedimento definido em outro processo. Desta forma, temos que reservar uma área de memória para dados locais suficientemente grande para acomodar a maior área de dados utilizada (figura 4.5).

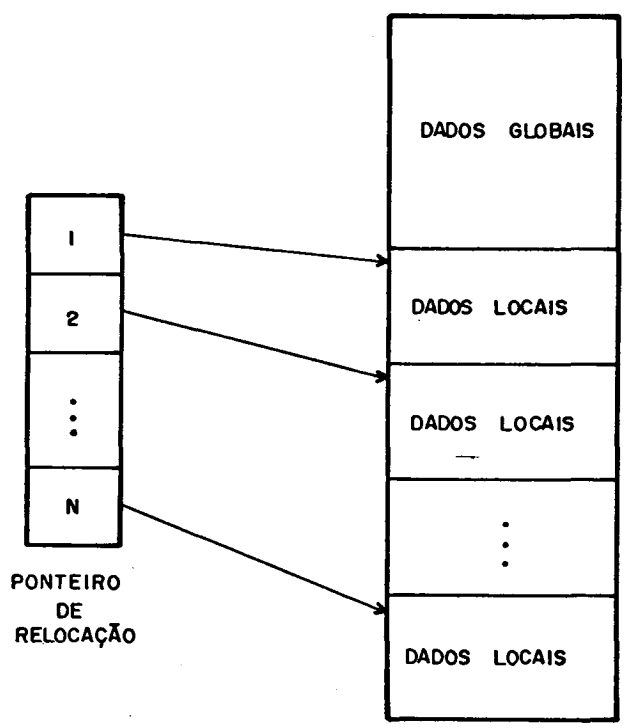


Figura 4.5 Estrutura dos dados

Um processo executa dois tipos de operações: o seu comando inicial e as chamadas remotas feitas pelos outros processos. Isto é feito intercalando a execução do comando inicial e as

chamadas remotas. O comando inicial e as chamadas remotas assumirão três estados durante a sua existência que são: EXECUTANDO, SUSPENSO e INATIVO. Um comando está no estado EXECUTANDO quando ele possui o processador operador, no estado SUSPENSO quando ele cede o processador operador para que outro comando seja executado e no estado INATIVO quando ele termina ou ainda não foi chamado. A figura 4.6 mostra o diagrama de transição de estado de um comando.

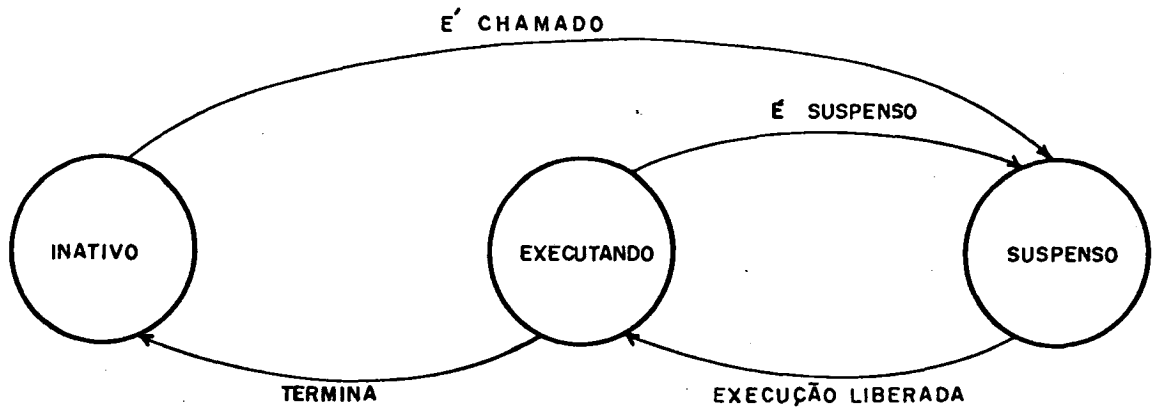


Figura 4.6 Transição de estado de um Comando

4.5 ESTRUTURA DO CONTROLADOR DO NÚCLEO

Para que estes Comandos sejam executados é preciso que sejam intercaladas as seguintes operações: *INSPECIONA ENTRADA, SAÍDA, SELECIONA EXECUÇÃO* e *EXECUÇÃO*. Estas operações têm o seguinte significado:

Inspeciona Entrada - verifica se há algum pedido de execução de comando para este processador operador.

Saída - Envia o resultado de uma chamada remota de procedimento.

Seleciona Execução - Determina o próximo comando cuja execução deva ser iniciada ou retomada, caso exista.

Execução - Executa o comando selecionado até que termine ou espere que alguma condição torne-se verdadeira.

A Figura 4.7 mostra a estrutura do controlador do núcleo.

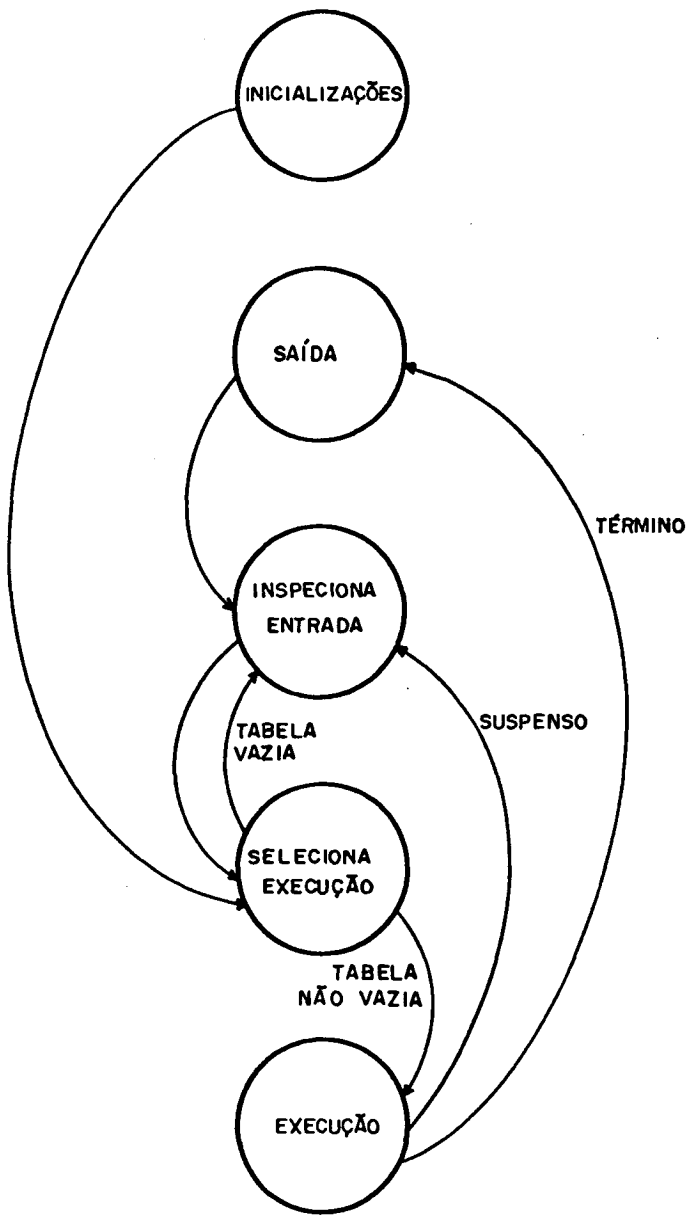


Figura 4.7 Controlador do Núcleo

Numa chamada remota de procedimento o processo cliente faz uma requisição para execução de um procedimento externo e fica bloqueado esperando o resultado. Como o cliente e o servidor são executados em processadores operadores diferentes pode acontecer uma falha no sistema de comunicação ou em um processador operador. Desta forma o cliente pode ficar esperando um resultado indefinidamente. Podemos ter ainda, por exemplo, o caso de um processo A que é suspenso à espera de que uma condição torne-se verdadeira e esta condição só pode ser mudada por uma chamada de um processo B. Se o processo B por sua vez for suspenso à espera de que uma condição torne-se verdadeira e esta condição só pode ser mudada por uma chamada do processo A, um processo ficará esperando pelo outro indefinidamente. Quando isto acontece dizemos que ocorreu um intertravamento dos processsos ou deadlock.

A confiabilidade não pode ser obtida sem o uso de mecanismos deselegantes que, passado um tempo estipulado, tomam providências caso uma das situações expostas acima aconteça |25|, |26|, |27|.

4.5.1 PRIMITIVA ENVIA

Uma mensagem é enviada de um processo para outro através de operações de saída usando uma porta de entrada/saída serial.

A primitiva bloqueada ENVIA, também chamada de *ENVIA SÍNCRONO*, tem a seguinte estrutura:

ENVIA (processo-destino, mensagem, VAR estado)

O processo emissor não pode proseguir até que o processo receptor confirme ter recebido a mensagem. O parâmetro "processo-destino" é o identificador do processador operador para onde a mensagem deve ser enviada, o parâmetro "mensagem" é um conjunto de bytes que constituem a informação que deve ser transmitida e "estado" retorna um valor booleano que, no caso de ser verdadeiro, significa que a mensagem foi enviada e, caso contrário, que a operação não obteve sucesso mesmo depois de várias tentativas. Neste último caso o NÚCLEO deve enviar mensagens alertando sobre o mau funcionamento do sistema.

4.5.2 PRIMITIVA RECEBE

Uma mensagem é recebida de outro processo através de operações de entrada usando uma porta de entrada/saída serial.

A primitiva bloqueada RECEBE, também chamada de RECEBE SÍNCRONO, tem a seguinte estrutura:

RECEBE (VAR mensagem, VAR estado, tempo-resposta)

O processo receptor não pode proseguir até que o processo emissor envie uma mensagem ou até que o tempo-resposta seja esgotado. O parâmetro "mensagem" é um conjunto de bytes que constituem a informação recebida e "estado" retorna um valor booleano que, no caso de ser verdadeiro, significa que a mensagem foi recebida e, caso contrário, que passado o "tempo-resposta" nenhuma mensagem foi recebida.

4.5.3 PRIMITIVA CHAMADA REMOTA DE PROCEDIMENTO

A primitiva CHAMADA REMOTA DE PROCEDIMENTO (CRP) é ativada durante a fase de EXECUÇÃO quando o processo faz uma referência a um procedimento externo. A implementação de uma primitiva CRP envolve o envio pelo cliente de uma requisição com uma mensagem para o servidor apropriado, em seguida a recepção da resposta do servidor. O servidor recebe o pedido de execução de um procedimento, executa o trabalho e envia o resultado para o cliente.

A CHAMADA REMOTA DE PROCEDIMENTO tem a seguinte estrutura:

CHAMADA-REMOTA (servidor, cliente, procedimento, parâmetros-entrada, VAR resultado, VAR estadoCRP, tempo-resposta)

O "servidor" especifica o processador operador que atende a CRP, "procedimento" é a especificação do procedimento a ser executado, "parâmetros-entrada" são os parâmetros pedidos pelo procedimento que vai ser executado remotamente, "resultado" recebe os parâmetros resultados enviados pelo servidor, "estadoCRP" retorna o estado da CRP e "tempo-resposta" especifica o tempo que o cliente poderá esperar por uma resposta. O "estadoCRP" tem, para a chamada remota de procedimento, o seguinte significado:

estadoCRP = executado: O procedimento especificado foi executado pelo servidor e as respostas estão no "resultado".

estadoCRP = não recebido: Não existe o processador operador especificado ou este processador operador está com defeito ou o Sistema de Comunicação está com defeito.

estadoCRP = não executado: O processador operador aceitou a chamada mas o tempo-resposta expirou e não chegou o resultado da execução remota.

O formato da mensagem na chamada do procedimento remoto e no retorno do resultado está especificado na figura 4.8.

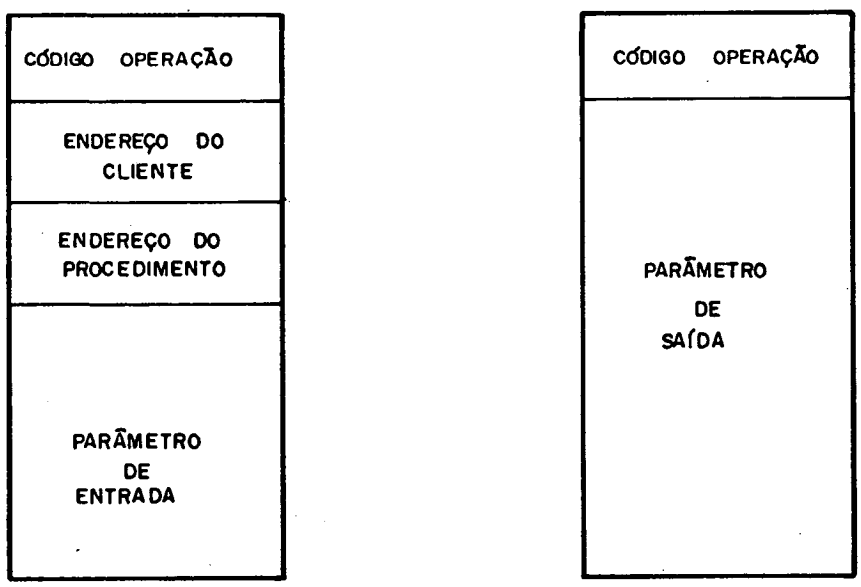


Figura 4.8 Formato da mensagem na CRP

O código de operação tem o seguinte significado:

Código Operação = CRP: Significa que esta mensagem é uma chamada remota de procedimento.

Código Operação = RES: Significa que esta mensagem é o resultado de uma chamada remota de procedimento.

```
TIPO msg-c = REGISTRO
                Código-Operação: byte;
                Endereço-Cliente: byte;
                Endereço-Procedimento: ... ;
                parâmetro: ... ;
             FIM
```

```
msg-s = REGISTRO
          Código-Operação: byte;
          parâmetro: ... ;
        FIM
```

Discutimos a seguir a implementação da Chamada Remota de Procedimento prevendo a ocorrência de possíveis falhas.

```
VAR mensagem: msg-c;
    result: msg-s;
{ (* montagem da estrutura da mensagem *)
  mensagem.Código-de-Operação:= CRP;
  mensagem.Endereço-Cliente:= ...
  mensagem.Endereço-procedimento:= ...
  mensagem.parâmetro-entrada:= ...
  (* inicializações *)
  retransmissão:=0;
  feito:= FALSO;
  (* cliente solicita a execução remota do procedimento *)
  REPITA
     ENVIA (servidor,mensagem,estadoE);
     SE estadoE
     ENTÃO feito:= VERDADEIRO
     SENÃO { retransmissão:=retransmissão + 1;
             SE retransmissão = max
             ENTÃO { estadoCRP:= não recebido;
                     feito:= VERDADEIRO }}
  ATÉ QUE feito;

  (* cliente espera o resultado da CRP *)
  feito := FALSO;
```

```
SE estadoE
ENTÃO
     REPITA
          RECEBE (result,estadoR,tempo-resposta);
          SE estadoR
          ENTÃO SE result.Código-Operação = RES
                ENTÃO { resultado:= result.parâmetro;
                        feito:= VERDADEIRO;
                        estadoCRP:= executado }
                SENÃO Colocar-CRP-Tabela
          SENÃO { estado-CRP:= não executado;
                  feito:= VERDADEIRO }
     ATÉ QUE feito;              }
```

4.6 OPERAÇÕES DO NÚCLEO

4.6.1 INSPECIONA ENTRADA

A operação Inspeciona Entrada executa a primitiva RECEBE e se uma mensagem for recebida ela deve ser transferida do buffer de recepção para a Tabela Descritora de Comandos.

```
{  RECEBE (buffer,estadoR,tempo-resposta);
   SE estadoR
   ENTÃO Colocar-CRP-Tabela      }
```

4.6.2 SAÍDA

Quando uma chamada remota está em EXECUÇÃO e termina, o resultado desta chamada deve ser enviado para o processo cliente. Isto é feito pela operação SAÍDA que carrega os

parâmetros de saída no buffer e executa a primitiva ENVIA.

```
VAR buffer: msg-s
{  buffer.Código-Operação:= RES;
   buffer.parâmetros:= ... ;
   cliente:= c-cor;
   ENVIA (cliente,buffer,estadoE);
   SE NÃO estadoE
   ENTÃO Erro: cliente não recebeu o resultado da CRP;
   Tabela-de-Processos.estado|id|:= FALSO        }
```

4.6.3 SELECIONA EXECUÇÃO

A operação Seleciona Execução tem que determinar o próximo Comando que irá executar e restaurar o ambiente volátil deste Comando. Se não existir nenhum processo para ser executado a variável booleana Tabela-Vazia retorna com o valor VERDADEIRO.

```
{  k:=0;
   p-cor:= p-cor MOD N + 1;
   ENQUANTO NÃO Tabela-de-Processos|p-cor|.estado E (k<N)
   FAÇA { p-cor:= p-cor MOD N + 1;
          k:=k+1 }
   SE k>=N
   ENTÃO Tabela.Vazia:= VERDADEIRO
   SENÃO Ambiente-Volátil:=Tabela-de-Processos|p-cor|     }
```

4.6.4 EXECUÇÃO

Quando a operação Seleciona Execução troca o contador de programa temos o inicio da execução de um novo Comando. Este Comando executa até que termine ou espere que alguma condição torne-se verdadeira. Se uma chamada remota de procedimento for feita então é chamada o procedimento CHAMADA-REMOTA e se a execução de uma chamada remota terminar então é chamado o procedimento SAÍDA.

4.6.5 INICIALIZAÇÕES

Toda inicialização é comandada através de um programa que executa em um dos processadores operadores.

Este programa tem a função de sincronizar o inicio da execução do programa distribuído. O processo destinado a comandar a fase de inicialização será chamado de processo mestre, por sua vez os outros processos serão chamados de processos escravos.

Os processos escravos fazem as inicializações necessárias e permanecem esperando até que o processo mestre ordene o início da execução. O processo mestre faz suas inicializações e em seguida passa a enviar mensagens para todos os processos escravos informando-os de que podem iniciar sua execução. Se algum processo escravo não receber a mensagem o processo mestre acusará o problema.

A inicialização nos processos escravos é feita da seguinte forma:

```
{  ... inicializações ... ;

   acabou:= FALSO;
   ENQUANTO NÃO acabou
   FAÇA { estadoR:=FALSO;
          ENQUANTO NÃO estadoR
          FAÇA RECEBE (buffer,estadoR,tempo-resposta);
          SE buffer.Código-Operação = INICIALIZAÇÃO
          ENTÃO acabou:= VERDADEIRO } ;

   atrasa(t); (* tempo para o processo mestre enviar  todas  as
mensagens *)
                                                          }
```

A inicialização no processo mestre é feita da seguinte forma:

```
{  ... inicializações ... ;

   Erro:= FALSO;
   I:= 1;
   ENQUANTO I <= N e NÃO Erro
   FAÇA { SE I <> N-op
          THEN { ENVIA (I,mensagem,estadoE);
                 SE NÃO estadoE
                 ENTÃO Erro:= VERDADEIRO;
                 I:= I + 1                    } };

   SE Erro
   ENTÃO ... o processador op. I não recebe mensagem ...   }
```

O programa distribuído fica gravado em diversas EPROMs e começa a ser executado quando o processador operador for ligado. Desta forma devemos ligar primeiro os processadores operadores que executam os processos servidores na fase de inicialização e só então ligarmos o procesador operador que executa o processo mestre.

Na fase de depuração do programa distribuído fica inviável gravar as EPROMs a cada alteração no programa. Pode-se usar um programa para receber os códigos objetos desejados e carregá-los na posição de memória adequada e um outro programa para enviar a todos os processadores operadores o código objeto alterado. Estes programas são executados antes do programa distribuído que começa com a execução das inicializações nos diversos processadores operadores.

4.7 IMPLEMENTAÇÃO

O Núcleo Básico na forma discutida neste capítulo foi implementado usando como processador operador um sistema de computação com central de processamento Z-80A, memória principal com 64 kbytes, interface para impressora, terminal de vídeo e unidades de disco flexível. Na comunicação do processador operador com o processador de comunicação utilizou-se uma interface serial.

Foi utilizado o sistema operacional CDOS (Cromemco Disk Operating System) que é uma extensão do sistema CP/M. Usou-se a linguagem Pascal e Assembler no desenvolvimento do Núcleo Básico.

O Sistema de Comunicação utilizado pelo Núcleo Básico está descrito no capítulo seguinte e as listagens estão no apêndice A2.

C A P Í T U L O 5

SUPORTE PARA IMPLEMENTAÇÃO DO NÚCLEO BÁSICO

5.1. INTRODUÇÃO

O Núcleo Básico para um Sistema Distribuído de Tempo Real necessita de mecanismos que façam a transmissão e recepção de mensagens entre os processadores operadores. Como já foi visto no capítulo 2 esta transferência de informações é feita através de um Sistema de Comunicação que pode ser do tipo ponto-a-ponto ou por difusão.

Sistemas de tempo real adaptam-se melhor ao sistema de comunicação do tipo difusão. No entanto, para permitir a experimentação das idéias desenvolvidas no capítulo 4, optou-se por utilizar o sistema de comunicação que se encontrava disponível.

Foi usado o Sistema de Comunicação ponto-a-ponto do Sistema de Desenvolvimento para Sistemas Distribuídos |28|,

|29|, |30|, |31|. Este Sistema de Desenvolvimento permite o desenvolvimento de projetos e estudos em Sistemas Distribuídos nas áreas de: topologia, protocolos, desempenho, linguagens, etc. O Sistema de Comunicação é composto por unidades básicas denominadas Processadores de Comunicação, que têm a finalidade de encaminhar as mensagens através deste sistema.

5.2. *SISTEMA DE DESENVOLVIMENTO PARA SISTEMAS DISTRIBUÍDOS*

O Sistema de Desenvolvimento para Sistemas Distribuídos tem um Sistema de Comunicação com topologia irregular, com conexão ponto-a-ponto e opera no modo "armazena e envia" (store-and-forward). O sistema é composto por processadores operadores ligados a processadores de comunicação. Os processadores de comunicação por sua vez são interligados por linhas de comunicação formando o Sistema de Comunicação. O processador operador realiza o processamento local, o pedido de execução remota e o atendimento de pedidos de execução remota.

Os principais fatos que resultaram nesta escolha foram:

- implementação de baixo custo
- possibilidade de utilizar qualquer processador comercial como processador operador
- possibilidade de utilização de quatro canais de comunicação serial "full-duplex"
- interface serial SIO-Z80 incorpora algumas características básicas do HDLC e permite comunicação assíncrona

A figura 5.1 mostra a configuração genérica do Sistema de Desenvolvimento para Sistemas Distribuídos.

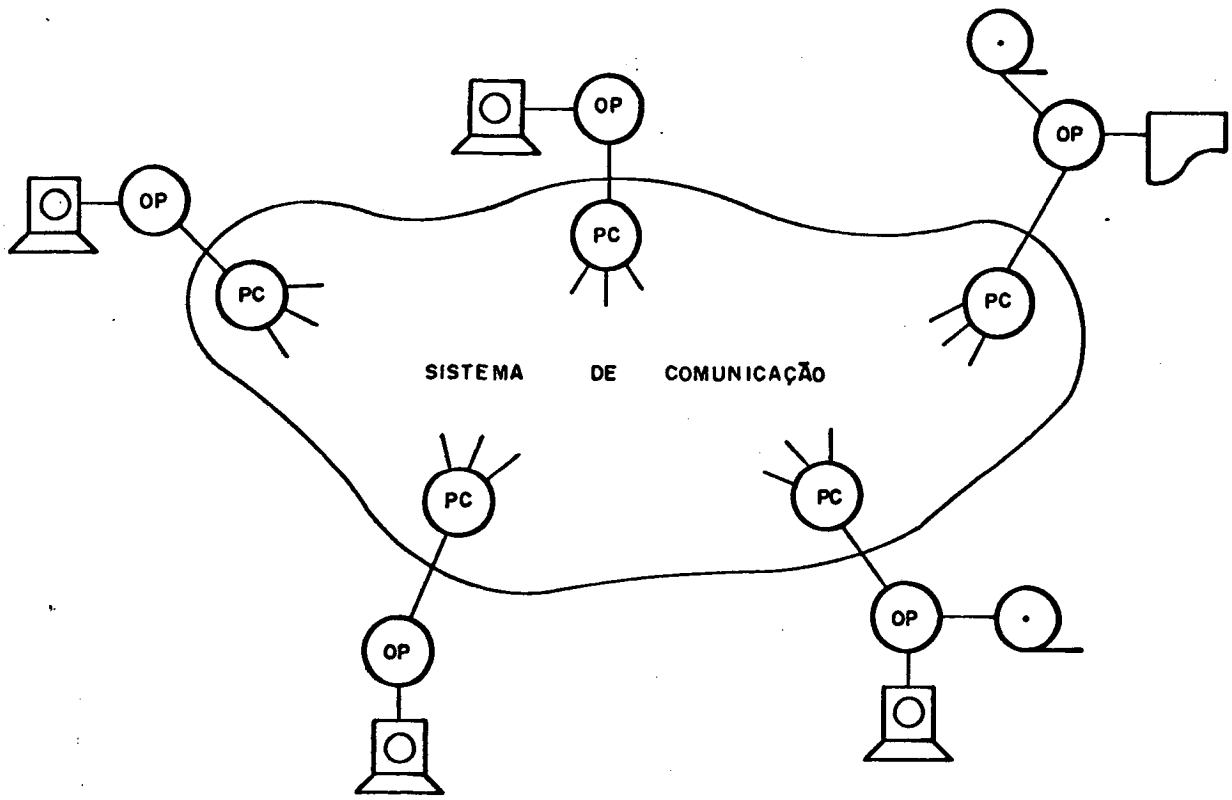


Figura 5.1 Sistema Distribuído Genérico

O processador de comunicação é composto por um microprocessador de oito bits Z-80 (Z-80 CPU Central Processing Controller) com duas interfaces com dois canais de comunicação serial "full-duplex" cada uma, duas "EPROMs" 2732 e 8 K de memória "RAM".

Dos quatro canais seriais "full-duplex" um é usado para a comunicação com o processador operador e três para a comunicação com os outros processadores de comunicação de acordo com a conveniência. A figura 5.2 mostra a arquitetura do processador de comunicação.

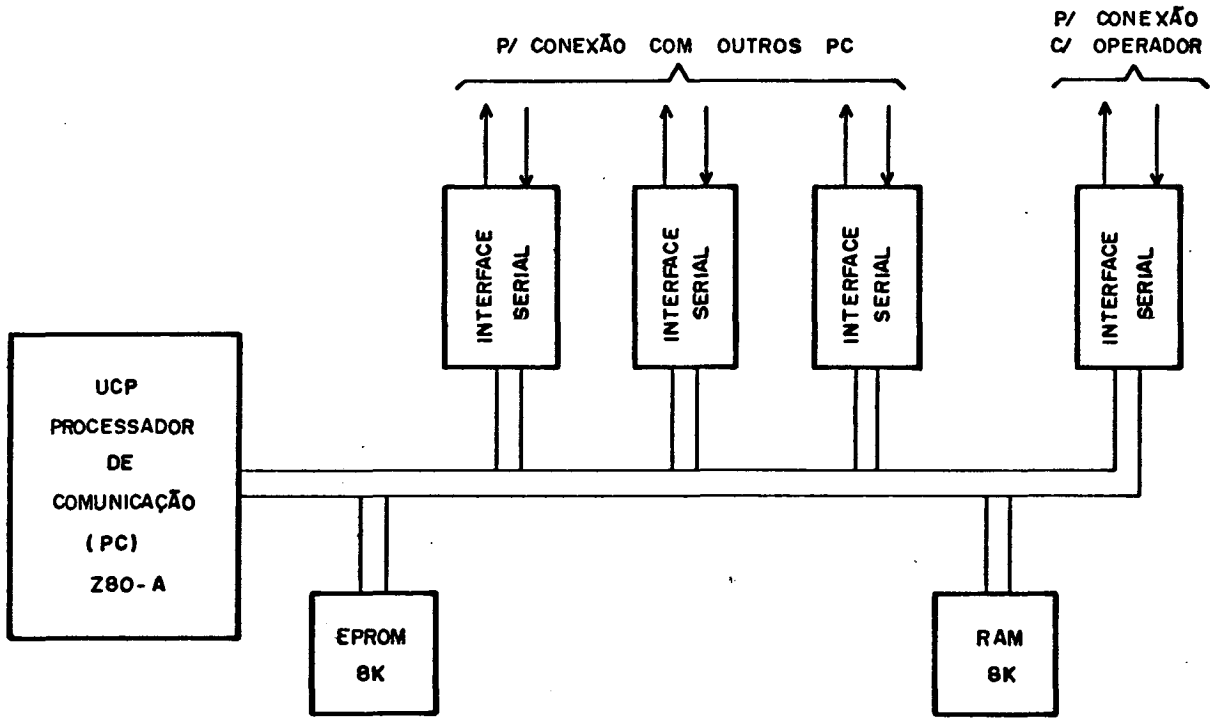


Figura 5.2 Arquitetura do Processador de Comunicação

5.3. *SUPERVISOR*

Num sistema distribuído a comunicação e sincronização de processos, localizados em processadores operadores diferentes, se realiza pela troca de mensagens através do Sistema de Comunicação. O Supervisor é um programa residente em cada processador de comunicação (PC) que tem por função encaminhar as mensagens através do Sistema de Comunicação, fazendo com que cheguem ao processador operador destino.

Na arquitetura proposta |30| todos os canais de comunicação são bi-direcionais, isto é, se o PC i pode transmitir mensagem para o PC j então o PC j pode transmitir para o PC i. A transferência de mensagens entre um PC e outro é feita através de uma interface serial (SIO-Z80). Cada

processador de comunicação tem três interfaces seriais, o que permite a troca de mensagens diretamente com três outros PCs do Sistema de Comunicação, e uma interface serial para a comunicação com o processador operador.

Quando um PC quer enviar uma mensagem para outro com quem não está diretamente ligado, terá que enviar a mensagem para um PC que esteja ligado diretamente ou através do qual seja possível chegar ao PC destino. Desta forma as mensagens fluem através do Sistema de Comunicação até chegar ao PC destino.

O Supervisor é o programa que faz com que estas mensagens fluam pelo Sistema de Comunicação, alternando recepção e transmissão, e sincronizando esta atuação para que nenhuma mensagem seja perdida. O supervisor testa também se a mensagem foi recebida sem erro, em caso positivo prossegue com o encaminhamento da mensagem, caso contrário pede nova transmissão.

5.4. TRANSMISSÃO E RECEPÇÃO

Os procedimentos de recepção e transmissão |31| têm como parâmetros as variáveis NBYTE, INIBT, INIBR, FIMBR, ESTADO-SIO com os seguintes significados:

NBYTE - Número de bytes a serem transmitidos

INIBT - Início do buffer de transmissão

INIBR - Início do buffer de recepção

FIMBR - Fim do buffer de recepção

ESTADO-SIO:

BIT	FUNÇÃO
0	Transmissor Ativo
1	Mensagem Completada
2	Transmissão Abortada
3	Receptor Ativo
4	Erro de OVERRUN
5	Erro de CRC
6	Fim de QUADRO
7	Detectado aborto na recepção

Para fazer uma transmissão o supervisor inicializa a variável NBYTE com o número de bytes que serão transmitidos e a variável INIBIT com o início do buffer a ser transmitido, e chama um procedimento de transmissão. O procedimento inicializa todos os seus ponteiros e devolve o controle para o supervisor. Em seguida, de acordo com a necessidade, atende interrupções até que todo o QUADRO seja transmitido. Neste instante fica assinalado na variável ESTADO-SIO que a mensagem foi completada.

A recepção ocorre de maneira semelhante e ao seu término a variável ESTADO-SIO assinala FIM DO QUADRO e as variáveis INIBR e FIMBR indicam respectivamente o início e o fim do QUADRO recebido.

5.5. ESTRUTURA DO PROCESSADOR DE COMUNICAÇÃO

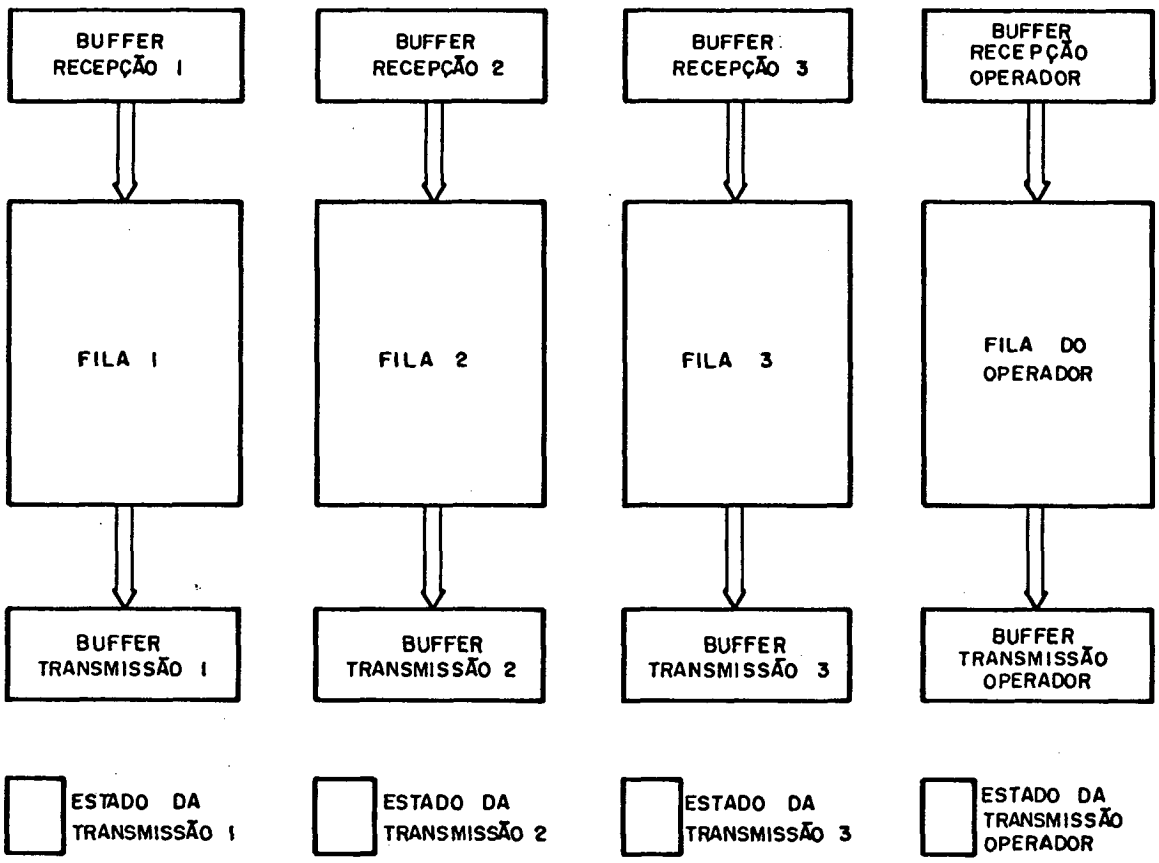


Figura 5.3 Estrutura das filas do PC

Uma mensagem recebida pelo supervisor é encaminhada para uma de suas quatro filas sendo três filas de espera de transmissão para outro PC e uma de transmissão para o processador operador. A determinação da fila onde a mensagem deve ser colocada para ser transmitida é feita por uma rotina de determinação de ROTA. Esta rotina consulta uma matriz "DISTÂNCIA ENTRE PCs", através da qual é possivel determinar o próximo PC para onde a mensagem deve ser transmitida [32].

Para que não tenhamos que fazer constantes movimentos do buffer para a fila e da fila para o buffer, a fila foi implementada com a estrutura de uma lista encadeada. Assim, por exemplo, a transferência do buffer para a fila consiste

simplesmente na colocação do buffer como um elemento da lista encadeada e definição de uma nova área de buffer. A nova área de buffer vem de uma lista encadeada de posições disponíveis para as quatro filas.

Apesar de as filas usarem a mesma área de posições disponíveis não será permitido que uma fila use toda a área disponível, o que em caso de falha de um PC ou um processador operador provocaria o bloqueio do processador de comunicação. Foi garantido um tamanho mínimo para cada uma das filas.

A área de armazenamento nas filas é igual para todo quadro e igual ao tamanho do maior quadro que pode ser aceito, mas é feita a transmissão somente dos bytes do quadro ignorando-se o restante.

As filas guardam, além dos quadros, o endereço do próximo quadro e o tamanho deste quadro.

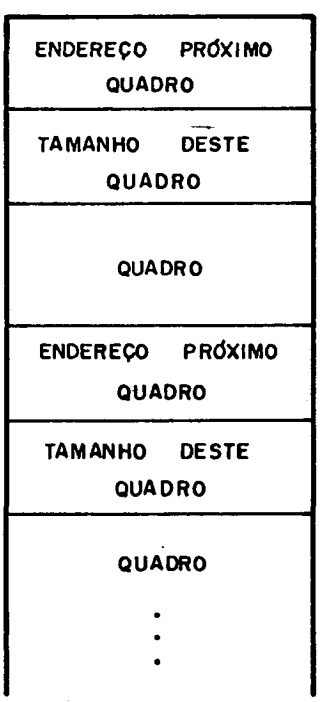
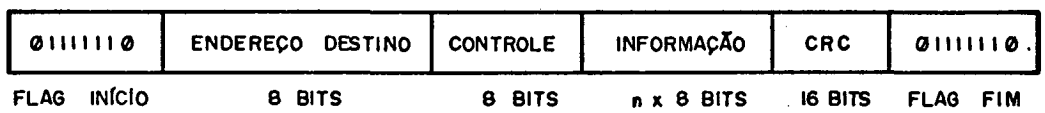


Figura 5.4 Organização das filas

5.6. PROTOCOLO

O protocolo implementado é uma simplificação do protocolo HDLC (High Level Data Link Control) e a interface SIO-Z80 oferece facilidades para o seu tratamento.

QUADRO COM INFORMAÇÃO



QUADRO SEM INFORMAÇÃO (CONTROLE DO SUPERVISOR)

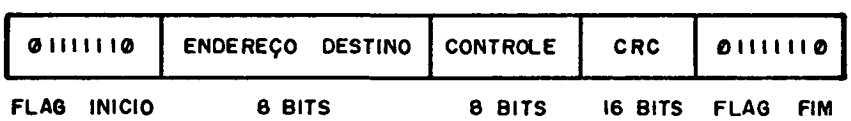


Figura 5.5 Protocolo de comunicação

É feita a seguir a descrição de cada um dos campos que formam a mensagem:

Flag início - Indica o início de um novo quadro

ENDEREÇO DESTINO - Endereço do processador operador destino para onde o quadro deve ser transmitido.

CONTROLE - Especifica o tipo do quadro que pode ser igual a :

1: Informação - Indica que este quadro carrega uma informação

2: Transmissão Rejeitada - Indica que a última transmissão foi recebida com erro, e que deve ser transmitido novamente o último quadro

3: Recebido e Liberado - Indica que a última transmissão

foi recebida corretamente e uma nova transmissão pode ser feita

4: Recebido e Não Liberado - Indica que a última transmissão foi recebida corretamente, mas não existe espaço na fila e portanto não pode ser feita outra transmissão

5: Inicialização - Indica que o Sistema de Comunicação vai ser inicializado ou reinicializado

6: Reconfiguração - Indica que o Sistema de Comunicação vai ser reconfigurado por falha ou por congestionamento de alguma rota

INFORMAÇÃO - Este campo é o texto da mensagem

CRC - Código de Redundância Cíclica

Flag Fim - Indica o fim do quadro

5.7. *ESTRUTURA DE FUNCIONAMENTO DO SISTEMA DE COMUNICAÇÃO*

O estado da transmissão é uma variável que pode assumir um dos seguintes valores:

1. Recebido e Liberado

Significa que a última transmissão realizada foi recebida corretamente e existe espaço na fila de recepção para receber a mensagem transmitida.

2. Recebido e Não Liberado

Significa que a última transmissão realizada foi recebida corretamente mas não existe espaço na fila de recepção para esta mensagem.

3. Esperando Confirmação

Significa que a transmissão foi realizada e o supervisor está esperando o reconhecimento.

A função da variável estado-transmissão é sincronizar a transferência de mensagens entre PCs, assinalando o estado atual de um módulo TRANSMISSÃO/RECEPÇÃO do supervisor. A figura 5.6 mostra o diagrama de estado deste módulo.

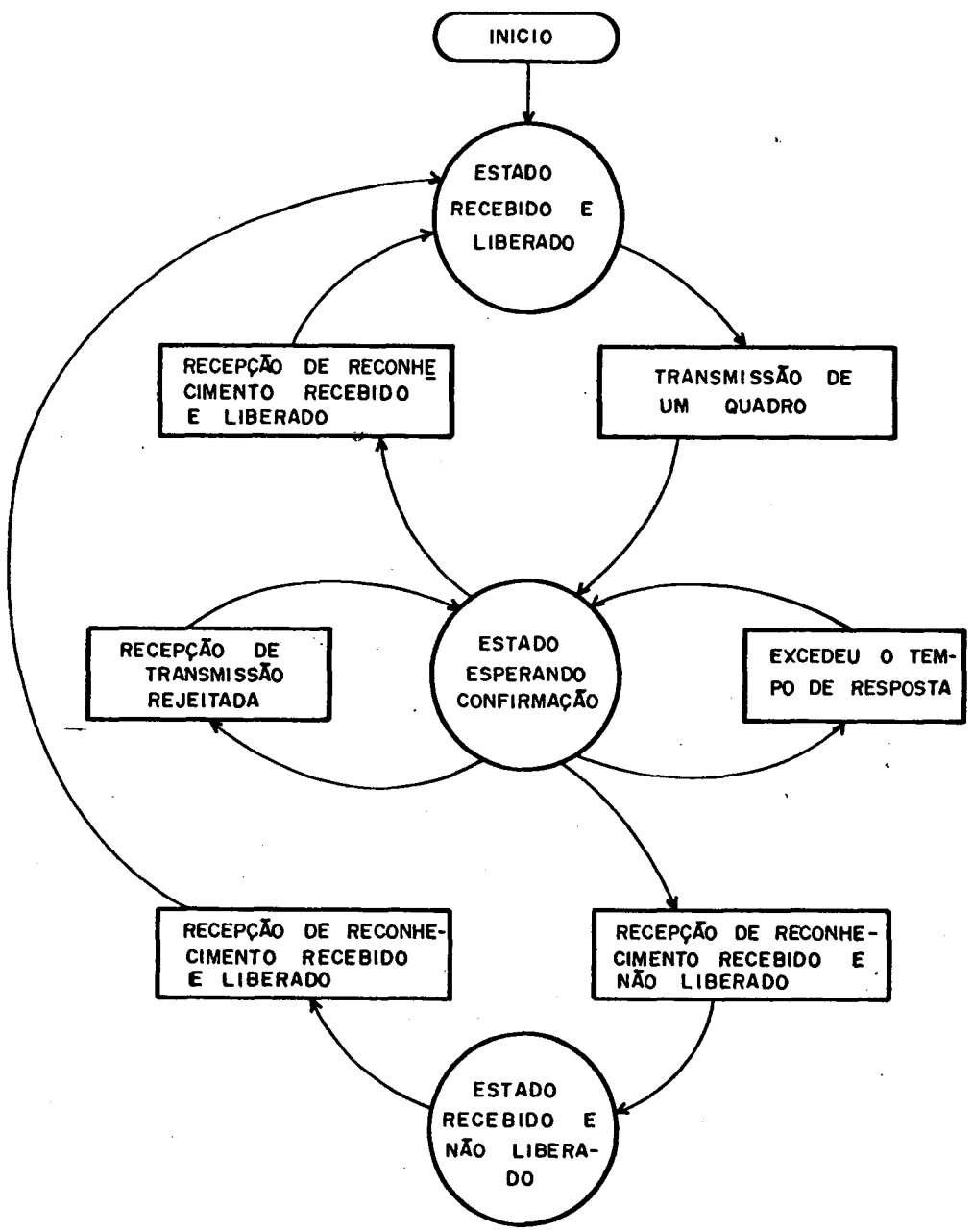


Figura 5.6 Diagrama de estado

O módulo estando no estado RECEBIDO E LIBERADO poderá fazer uma transmissão de um quadro da fila de espera e passar para o estado ESPERANDO CONFIRMAÇÃO até que ocorra a recepção de um reconhecimento.

Do estado ESPERANDO CONFIRMAÇÃO o módulo passa para o estado RECEBIDO e LIBERADO caso ocorra a recepção de um reconhecimento "recebido e liberado" ou para o estado RECEBIDO e NÃO LIBERADO com a recepção de um reconhecimento "recebido e não liberado". Ainda neste estado é possível receber uma mensagem de "transmissão rejeitada" ou ultrapassar o tempo de resposta estipulado, nestes casos é feita nova transmissão e o estado não é alterado.

Do estado RECEBIDO e NÃO LIBERADO no caso de recepção de um reconhecimento "recebido e liberado" o módulo passa para o estado RECEBIDO e LIBERADO.

5.8. COMUNICAÇÃO ENTRE PO E PC

Um dos quatro canais seriais das interfaces SIO-Z80 é reservado para a comunicação entre o processador operador e o processador de comunicação. Se o processador operador possuir a mesma interface que o processador de comunicação, ou uma equivalente, poderá ser usada a estrutura utilizada na comunicação PC-PC. Desta forma tanto a comunicação dentro do Sistema de Comunicação, assim como a comunicação entre o Sistema de Comunicação e o processador operador terão o mesmo padrão.

Como a maioria dos computadores comerciais tem uma interface serial que opera de acordo com a norma RS-232, foi desenvolvido um protocolo assíncrono segundo esta norma usando a mesma interface serial SIO-Z80 do processador de comunicação.

Os procedimentos de recepção assíncrona de uma cadeia de bytes têm a seguinte forma:

RECEBE-A(NBYTE,Var Buffer-R,tempo-resposta)

onde:

NBYTE: Número de bytes a ser recebido

Buffer-R: Buffer de recepção

tempo-resposta: Intervalo de tempo que o procedimento pode esperar para que a recepção aconteça

O procedimento de transmissão assíncrona de uma cadeia de bytes tem a seguinte forma:

TRANSMITE-A(NBYTE,Var Buffer-T)

onde:

NBYTE: Número de bytes a ser transmitido

Buffer-T: Buffer de transmissão

O processador de comunicação fica varrendo os quatro canais de Entrada/Saída fazendo recepções e transmissões. Desta forma o processador de comunicação não pode ficar somente à espera de que o processsador operador faça uma transmissão, mas se ela for feita sem que o processador de comunicação esteja esperando, esta transmissão será perdida. Da mesma forma o processador operador não pode ficar somente esperando uma transmissão do processador de comunicação, mas tem também que

executar os seus processos.

É preciso antes de fazer uma transmissão ter certeza que o destinatário está pronto para receber, seja ele o processador operador ou processador de comunicação. A solução é ficar enviando uma mensagem para o destinatário até que ele responda que está pronto para receber uma mensagem.

```
Procedimento RECEPÇÃO(Buffer-R,tempo-resposta,Var recebeu);
{NBYTE:= N-pequeno;
 RECEBE-A(NBYTE,B-aux-R,tempo-resposta);
 SE B-aux-R = Código quero transmitir
 ENTÃO { B-aux-T:= Código pode transmitir;
         TRANSMITE-A(NBYTE,B-aux-T);
         NBYTE:= N-grande;
         RECEBE-A(NBYTE,Buffer-R,tempo-resposta);
         recebeu:=VERDADEIRO              }
 SENÃO recebeu:=FALSO    }
```

```
Procedimento TRANSMISSÃO(Buffer-T, Var transmitiu);
{NBYTE:= N-pequeno;
 transmitiu:= FALSO;
 N-tentativas:= 0;
 ENQUANTO NÃO transmitiu E N-tentativas < Nt-max
 FAÇA { N-tentativas:= N-tentativas + 1;
        TRANSMITE-A(NBYTE,B-aux-T);
        RECEBE-A(NBYTE,B-aux-R,tempo-max);
        SE B-aux-R = Código pode transmitir
        ENTÃO transmitiu:= VERDADEIRO    };
```

```
SE transmitiu
ENTÃO { NBYTE:= N-grande;
        TRANSMITE-A(NBYTE,Buffer-T)   }      }
```

O problema agora é conseguir a comunicação entre o PO e o PO livre de erros, já que a linha física de transmissão que faz esta interconexão está sujeita a diversos tipos de interferências e mal funcionamento que resultam em erros de transmissão. Estes erros podem ser a alteração, perda ou duplicação da informação.

Nosso objetivo é conseguir um canal virtual capaz de transmitir informações de forma confiável apesar da linha física existente poder introduzir erros na transmissão.

A linha de transmissão pode apresentar erros que causam a alteração ou perda completa da informação transmitida. Temos então a necessidade de informar ao emissor que a informação foi adequadamente recebida pelo receptor. Este mecanismo é chamado de confirmação (ACK - "acknowledgment").

Se um quadro for perdido não haverá confirmação e o emissor ficará esperando eternamente uma confirmação. Uma solução é a utilização de um mecanismo de temporização no emissor associado com o mecanismo de confirmação. Se o emissor não receber a confirmação dentro de um tempo estipulado, ele deverá enviar novamente o quadro não confirmado.

A solução apresentada permite que aconteça o seguinte caso: o receptor recebe corretamente um quadro, e envia sua confirmação. Se a confirmação se perder o emissor vai

retransmitir o quadro já recebido e aceito. Teremos então a duplicação de um quadro.

O erro exposto pode ser superado se o receptor tiver a capacidade de distingüir entre quadros novos e quadros retransmitidos. Isto pode ser feito usando um número de seqüência de quadros. O emissor mantém o número do último quadro enviado e não confirmado. O receptor mantém o número do último quadro recebido. Ao receber um quadro, o receptor compara o número de seqüência do quadro recebido com o número que mantém armazenado. Se forem diferentes, é um novo quadro e ele confirma o recebimento. Se forem iguais, é um quadro duplicado; o receptor ignora este quadro e envia novamente a confirmação.

A figura 5.7 mostra o protocolo utilizado na comunicação PO-PC.

ENDEREÇO DESTINO	CONTROLE	NÚMERO DE SEQUENCIA	ENDEREÇO EMISSOR	INFORMAÇÃO	VERIFICAÇÃO DE ERROS
8 BITS	8 BITS	8 BITS	8 BITS	n x 8 BITS	8 BITS

Figura 5.7 Protocolo de comunicação PO-PC

A temporização pode influir no bom funcionamento do protocolo. Suponhamos o caso em que ocorra a temporização enquanto a confirmação estava a caminho. O emissor envia novamente o quadro e recebe a confirmação do quadro anterior,

supondo que seja do último quadro enviado. O receptor recebe o quadro duplicado que é rejeitado e faz a sua confirmação. Se neste momento o emissor envia um quadro que é perdido e em seguida recebe a confirmação do quadro duplicado como se fosse do quadro perdido, teremos então a perda de um quadro que não é detectada. A solução pode ser conseguida fazendo com que a confirmação traga o número de seqüência do quadro recebido.

```
CONST tam-msg = ... ;
TIPO msg = arranjo |1..tam-msg| de byte;
QUADRO = REGISTRO
            endereço: byte;
            controle: byte;
            seqüência: inteiro;
            end-emis: byte;
            info: msg;
         FIM;

Procedimento ENVIA(processo-destino,mensagem,Var estado);
{ quadro.endereço:= processo-destino;
  quadro.controle:= Código informação;
  quadro.seqüência:= número-seqüência;
  quadro.end-emis:= endereço-emissor;
  quadro.info:= mensagem;
  estado:= VERDADEIRO;

  TRANSMISSÃO(quadro,transmitiu);
  SE transmitiu
  ENTÃO { Acabou:= VERDADEIRO;
          N-tentativas:= 0;
```

```
            ENQUANTO not Acabou e N-tentativas < Nt-max
            FAÇA { RECEPÇÃO(B-aux-R,tempo-max,Recebeu);
                    SE Recebeu
                    ENTÃO    SE    B-aux-R.controle    =    Código
informação
                                ENTÃO    SE    B-aux-R.seqüência    =
 número-seqüência
                                        ENTÃO Acabou:=VERDADEIRO} ;
            N-tentativas:= N-tentativas + 1 };

    SE NÃO  Acabou
    ENTÃO estado:= FALSO
    SENÃO SE número-seqüência = seq-max
            ENTÃO número-seqüência:= número-seqüência + 1 }
 SENÃO estado:= FALSO }


Procedimento RECEBE(Var mensagem, Var estado, tempo-resposta);
{ estado:= VERDADEIRO;

  RECEPÇÃO(quadro,tempo-resposta,Recebeu);
  SE Recebeu
  ENTÃO { SE quadro.seqüência = n-seq-esperado
            ENTÃO { mensagem:= quadro.info;
                    SE n-seq-esperado = seq-max
                    ENTÃO número-seqüência:= 1
                    SENÃO                    número-seqüência:=
número-seqüência+1}
            SENÃO estado:=FALSO;
```

```
            B-aux-T.endereço:= quadro.end-emis;
            B-aux-T.controle:= Código confirmação;
            B-aux-T.seqüência:= quadro.seqüência;

            TRANSMISSÃO(B-aux-T, transmitiu); }
SENÃO estado:=FALSO }
```

CAPÍTULO 6

CONCLUSÃO

O Sistema de Comunicação ponto-a-ponto discutido no capítulo 5 tem desempenho proporcional ao número de processadores de comunicação. Na estrutura apresentada onde o processador de comunicação tem três interfaces seriais para comunicação com os outros processadores de comunicação, é possível ter quatro processadores de comunicação totalmente interligados. Isto é, todo processador de comunicação pode se comunicar diretamente com qualquer outro do Sistema de Comunicação. Se for usado um número maior de processadores deve-se procurar agrupá-los de tal forma que os processos que tenham um tempo crítico de comunicação estejam diretamente conectados. Se o problema não puder ser solucionado desta forma então ou aumenta-se o número de interfaces do Sistema de Comunicação ou será necessária a utilização de um outro Sistema de Comunicação do tipo "broadcast" (difusão).

O modelo de computação distribuída utilizado baseado em Distributed Processes de Brinch Hansen tem algumas limitações em certas circunstâncias.

O fato de cada processador executar um único processo torna a estrutura muito rígida e pode não explorar todo o potencial do Sistema Distribuído. As conseqüências deste fato são:

1 - Quanto ao Custo

Se o objetivo for obter um Sistema Distribuído de baixo custo então pode ser desejável que cada processador execute mais que um processo.

2 - Quanto à Flexibilidade

Nenhum processo novo pode ser introduzido no Sistema Distribuído sem a alteração da arquitetura do hardware, restringindo a possibilidade de expansão. Alguns sistemas podem requerer que o número de processos no sistema varie com a carga de trabalho do sistema.

3 - Quanto à Confiabilidade

Os problemas de confiabilidade são resolvidos com a introdução de módulos processador/memória redundantes no Sistema Distribuído. Quando uma falha ocorre o sistema é reconfigurado usando o módulo reserva. Se houver uma alocação rígida dos processos nos processadores a reconfiguração se torna mais difícil.

4 - Quanto à Generalidade

A generalidade do Sistema Distribuído é afetada pela rigidez da estrutura adaptando-se melhor a aplicações com um

pequeno número de processos fixos.

O Núcleo Básico para um Sistema Distribuído pode ser expandido para incorporar a execução de mais de um processo por processador. Cada processador operador executará chamadas externas para seus procedimentos e o comando inicial de cada um dos seus processos. Como cada processo ao fazer uma chamada remota fica bloqueado até obter seu resultado, isto não vai alterar a estrutura proposta. Mas o fato do processador operador executar mais que um processo precisa ser representado na Tabela Descritora de Comandos.

Uma solução é a inclusão de um campo de ligação na tabela descritora que indique qual o próximo processo a ser executado. A figura 6.1 mostra a extensão da tabela descritora de comandos de um processador operador que executa três processos concorrentes.

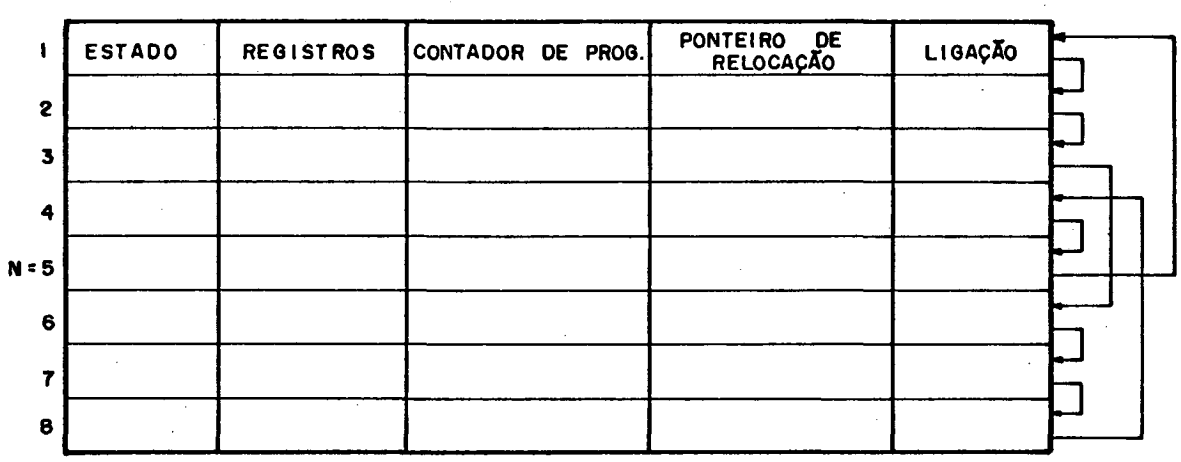


Figura 6.1 Tabela Descritora de Comandos

Esta expansão torna o Núcleo Básico mais genérico e possibilita a execução de um ou mais processos no mesmo processador operador, de acordo com a necessidade do momento. Entretanto, ao alocar dois processos em um mesmo processador operador deve-se tomar o cuidado de verificar se como conseqüência não teremos um aumento demasiado no tempo de resposta, um fator crítico em sistemas de tempo real.

A P Ê N D I C E A 1

LISTAGENS DO NÚCLEO BÁSICO PARA UM SISTEMA DISTRIBUÍDO DE TEMPO REAL

```
Stmt  Nest    Source Statement

   1     0    PROGRAM ENVIARECEBE;
   2     0
   3     0
   4     0    CONST CODIGOINFORMACAO=0; (* CONTROLE INFORMACAO *)
   5     1          NTMAX = 10;        (* NRO MAXIMO DE TENTATIVAS *)
   6     1          SEQMAX = 200;      (* NRO DE SEQUENCIA MAXIMO *)
   7     1          TAMMSG = 25;       (* TAMANHO DA MENSAGEM *)
   8     1          TAMP = 10;         (* TAMANHO DO PARAMETRO *)
   9     1          N = 2;             (* NRO DE PROCESSADORES OPER. *)
  10     1          ID = 1;            (* END. DESTE PROCESSADOR OPER. *
  11     1          CRP = 1;           (* CODIGO = CRP *)
  12     1          RES = 2;           (* CODIGO = RESPOSTA DA CRP *)
  13     1          MAX = 4;           (* NRO MAXIMO DE RETR. NA CRP *)
  14     1          EXECUTADO = 1;     (* CRP = EXECUTADO *)
  15     1          NAORECEBIDO = 2;   (* CRP = NAO RECEBIDO *)
  16     1          NAOEXECUTADO = 3;(* CRP = NAO EXECUTADO *)
  17     1
  18     1
  19     1    TYPE  MSG = ARRAY [1..TAMMSG] OF INTEGER;
  20     1          TIPOPAR = ARRAY [1..TAMP] OF INTEGER;
  21     1          DESCRITOR = RECORD
  22     1                         ESTADO:BOOLEAN;
  23     1                         AF,BC,DE,HL:INTEGER;
  24     1                         CONTADOR:INTEGER;
  25     1                         PONTEIRO:INTEGER
  26     1                       END;
  27     1          MSGC = RECORD
  28     1                    CONTROLE:BYTE;
  29     1                    ENDERECO:BYTE;
  30     1                    SEQUENCIA:INTEGER;
  31     1
  32     1                    INFO:MSG
  33     1                  END;
  34     1
  35     1
  36     1    VAR     I,J:INTEGER;
  37     1            QUADRO : MSGC;
  38     1            TRANSTIU,RECEBEU:BOOLEAN;
  39     1            NUMEROSEQUENCIA:INTEGER;
  40     1            NSEQREC:INTEGER;
  41     1            SEMPRE:BOOLEAN;
  42     1            TABELAP:ARRAY[1..N] OF DESCRITOR;
  43     1            PCOR,XBC,XDE,XHL,XAF,XCONTADOR,XPONTEIRO:INTEGER;
  44     1            TABVAZIA,TERMINO:BOOLEAN;
  45     1            ENDPR:INTEGER;
  46     1            ESTADOCRP:INTEGER;
  47     1            PARAMET:TIPOPAR;
  48     1
  49     1    PROCEDURE RECEPCAO;
  50     1    (* CHAMA A SUBROTIMA EM ASSEMBLER QUE FAZ A RECEPCAO *)
  51     1    CONST REC = 1444;  (* ENDERECO DA SUBROTINA RECEPCAO *)
  52     2          PIB = 1214;  (* ENDERECO DO BUFFER DE RECEPCAO *)
  53     2          TAM = 53;    (* TAMANHO DA MENSAGEM *)
  54     2          RCBEU = 1381;(* VARIAVEL QUE INDICA SE RECEBEU *)
```

```
Stmt  Nest    Source Statement

  55     2    BEGIN
  56     2        INLINE($C5   /                (* PUSH BC *)
  57     2               $F5   /                (* PUSH AF *)
  58     2               $E5   /                (* PUSH HL *)
  59     2               $D5   /                (* PUSH DE *)
  60     2               $DD   /                (* PUSH IX *)
  61     2               $E5   /
  62     2               $FD   /                (* PUSH IY *)
  63     2               $E5   /
  64     2
  65     2               $CD   / REC /          (* CALL REC *)
  66     2               $21   / PIB /          (* LD HL,PIB *)
  67     2               $11   / I /            (* LD DE,I *)
  68     2               $13   /                (* INC DE *)
  69     2               $13   /                (* INC DE *)
  70     2               $01   / TAM /          (* LD BC,TAM *)
  71     2               $ED   /                (* LDIR *)
  72     2               $B0   /
  73     2
  74     2               $3A   / RCBEU /        (* LD A,(RCBEU) *)
  75     2               $32   / RECEBEU /      (* LD (RECEBEU),A *)
  76     2
  77     2               $FD   /                (* POP IY *)
  78     2               $E1   /
  79     2               $DD   /                (* POP IX *)
  80     2               $E1   /
  81     2               $D1   /                (* POP DE *)
  82     2               $E1   /                (* POP HL *)
  83     2               $F1   /                (* POP AF *)
  84     2               $C1   )                (* POP BC *)
  85     2    END;
  86     1
  87     1
  88    -1    PROCEDURE TRANSMISSAO;
  89     1    (* CHAMA A SUBROTINA ASSEMBLER QUE FAZ A TRANSMISSAO *)
  90     1    CONST PIBT = 1300;    (* ENDERECO DO BUFFER DE TRANSM. *)
  91     2          TRTIU = 1382;   (* VARIAVEL INDICA SE TRANSMITIU *)
  92     2          TRANSM = 1715;  (* ENDERECO SUBROTINA DE TRANSM. *)
  93     2          TAM = 53;       (* TAMANHO DA MENSAGEM *)
  94     2    BEGIN
  95     2        INLINE($C5   /                (* PUSH BC *)
  96     2               $F5   /                (* PUSH AF *)
  97     2               $E5   /                (* PUSH HL *)
  98     2               $D5   /                (* PUSH DE *)
  99     2               $DD   /                (* PUSH IX *)
 100     2               $E5   /
 101     2               $FD   /                (* PUSH IY *)
 102     2               $E5   /
 103     2
 104     2               $21   / I /            (* LD HL,I *)
 105     2               $23   /                (* INC HL *)
 106     2               $23   /                (* INC HL *)
 107     2               $11   / PIBT /         (* LD DE,PIBT *)
 108     2               $01   / TAM /          (* LD BC,TAM *)
```

```
Stmt  Nest    Source Statement

 109     2                $ED   /               (* LDIR *)
 110     2                $B0   /
 111     2
 112     2                $CD   / TRANSM /      (* CALL TRANSM *)
 113     2
 114     2                $3A   / TRTIU /       (* LD A,(TRTIU) *)
 115     2                $32   / TRANSTIU /    (* LD (TRANSTIU),A *)
 116     2
 117     2                $FD   /               (* POP IY *)
 118     2                $E1   /
 119     2                $DD   /               (* POP IX *)
 120     2                $E1   /
 121     2                $D1   /               (* POP DE *)
 122     2                $E1   /               (* POP HL *)
 123     2                $F1   /               (* POP AF *)
 124     2                $C1   )               (* POP BC *)
 125     2    END;
 126     1
 127     1    PROCEDURE INICIALIZACAO;
 128     1    (* CHAMA A SUBROTINA ASSEMBLER QUE FAZ A INICIALIZACAO
 129     1       PARA A TRANSMISSAO E RECEPCAO *)
 130     1    CONST INICIO = 632;  (* END. SUBROTINA INICIALIZACAO *)
 131     2    BEGIN
 132     2        INLINE($C5   /               (* PUSH BC *)
 133     2               $F5   /               (* PUSH AF *)
 134     2               $E5   /               (* PUSH HL *)
 135     2               $D5   /               (* PUSH DE *)
 136     2               $DD   /               (* PUSH IX *)
 137     2               $E5   /
 138     2               $FD   /               (* PUSH IY *)
 139     2               $E5   /
 140     2
 141     2               $CD   / INICIO /      (* CALL INICIO *)
 142     2
 143     2               $FD   /               (* POP IY *)
 144     2               $E1   /
 145     2               $DD   /               (* POP IX *)
 146     2               $E1   /
 147     2               $D1   /               (* POP DE *)
 148     2               $E1   /               (* POP HL *)
 149     2               $F1   /               (* POP AF *)
 150     2               $C1     )             (* POP BC *)
 151     2    END;
 152     1
 153     1    PROCEDURE ENVIA(PROCESSODESTINO:BYTE;MENSAGEM:MSG;
 154     1              VAR ESTADO:BOOLEAN);
 155     1    VAR ACABOU:BOOLEAN;
 156     2        NTENTATIVAS:INTEGER;
 157     2    BEGIN
 158     2        QUADRO.ENDERECO:=PROCESSODESTINO;
 159     2        QUADRO.CONTROLE:=CODIGOINFORMACAO;
 160     2        QUADRO.SEQUENCIA:=NUMEROSEQUENCIA;
 161     2        QUADRO.INFO:=MENSAGEM;
 162     2        ESTADO:=TRUE;
```

```
Stmt  Nest    Source Statement

 163     2
 164     2        TRANSMISSAO;
 165     2
 166     2        IF TRANSTIU
 167     2        THEN BEGIN
 168     3               ACABOU:=FALSE;
 169     3               NTENTATIVAS:=0;
 170     3               WHILE NOT ACABOU AND (NTENTATIVAS <NTMAX)
 171     3               DO BEGIN
 172     4                     RECEPCAO;
 173     4                     IF RECEBEU
 174     4                     THEN IF QUADRO.CONTROLE =
 175     4                             CODIGOINFORMACAO
 176     4                          THEN IF QUADRO.SEQUENCIA =
 177     4                                  NUMEROSEQUENCIA
 178     4                               THEN ACABOU:=TRUE;
 179     4               NTENTATIVAS:=NTENTATIVAS+1
 180     4                    END;
 181     3
 182     3             IF NOT ACABOU
 183     3             THEN ESTADO:=FALSE
 184     3             ELSE IF NUMEROSEQUENCIA = SEQMAX
 185     3                  THEN NUMEROSEQUENCIA:=1
 186     3                  ELSE NUMEROSEQUENCIA:=NUMEROSEQUENCIA+1
 187     3              END
 188     3        ELSE ESTADO:=FALSE
 189     2    END;
 190     1
 191     1    PROCEDURE RECEBE(VAR MENSAGEM:MSG;VAR ESTADO:BOOLEAN;
 192     1              TEMPORESPOSTA:INTEGER);
 193     1    BEGIN
 194     2        ESTADO:=TRUE;
 195     2        RECEPCAO;
 196     2        IF RECEBEU
 197     2        THEN IF QUADRO.CONTROLE=CODIGOINFORMACAO
 198     2             THEN BEGIN
 199     3                  IF QUADRO.SEQUENCIA =NSEQREC
 200     3                  THEN BEGIN
 201     4                        MENSAGEM:=QUADRO.INFO;
 202     4                        IF NSEQREC = SEQMAX
 203     4                        THEN NSEQREC:=1
 204     4                        ELSE NSEQREC:=NSEQREC + 1
 205     4                       END
 206     4                  ELSE ESTADO:=FALSE;
 207     3
 208     3                  QUADRO.ENDERECO:=QUADRO.INFO[2];
 209     3                  QUADRO.CONTROLE:=CODIGOINFORMACAO;
 210     3
 211     3                  TRANSMISSAO;
 212     3                  END
 213     3             ELSE ESTADO:=FALSE
 214     2        ELSE ESTADO:=FALSE
 215     2    END;
 216     1
```

```
Stmt  Nest    Source Statement

 217     1    PROCEDURE SELECIONAEXECUCAO;
 218     1    VAR K:INTEGER;
 219     2    BEGIN
 220     2        TABVAZIA:=FALSE;
 221     2        K:=0;
 222     2        PCOR:=PCOR MOD N + 1;
 223     2        WHILE NOT TABELAP[PCOR].ESTADO AND (K<N)
 224     2        DO BEGIN
 225     3               PCOR:=PCOR MOD N + 1;
 226     3               K:=K+1
 227     3           END;
 228     2        IF K>=N
 229     2        THEN TABVAZIA:=TRUE
 230     2    END;
 231     1
 232     1    PROCEDURE EXECUCAO;
 233     1    BEGIN
 234     2        TERMINO:=FALSE;
 235     2        XBC:=TABELAP[PCOR].BC;
 236     2        XDE:=TABELAP[PCOR].DE;
 237     2        XHL:=TABELAP[PCOR].HL;
 238     2        XAF:=TABELAP[PCOR].AF;
 239     2        XCONTADOR:=TABELAP[PCOR].CONTADOR;
 240     2        INLINE($ED   /
 241     2               $4B   / XBC /          (* LD BC,(XBC) *)
 242     2               $ED   /
 243     2               $5B   / XDE /          (* LD DE,(XDE) *)
 244     2               $2A   / XAF /          (* LD HL,(XAF) *)
 245     2               $E5   /                (* PUSH HL *)
 246     2               $F1   /                (* POP AF *)
 247     2               $2A   / XCONTADOR /    (* LD HL,(XCONTADOR) *)
 248     2               $E5   /                (* PUSH HL *)
 249     2               $2A   / XHL )          (* LD HL,(XHL) *)
 250     2    END;
 251     1
 252     1    PROCEDURE SALVAAMBIENTE;
 253     1    BEGIN
 254     2        INLINE($22   / XHL /          (* LD (XHL),HL *)
 255     2               $ED   /
 256     2               $43   / XBC /          (* LD (XBC),BC *)
 257     2               $ED   /
 258     2               $53   / XDE /          (* LD (XDE),DE *)
 259     2               $F5   /                (* PUSH AF *)
 260     2               $E1   /                (* POP HL *)
 261     2               $22   / XAF );         (* LD (XAF),HL *)
 262     2        TABELAP[PCOR].BC:=XBC;
 263     2        TABELAP[PCOR].DE:=XDE;
 264     2        TABELAP[PCOR].HL:=XHL;
 265     2        TABELAP[PCOR].AF:=XAF;
 266     2    END;
 267     1
 268     1
 269     1
 270     1
```

```
Stmt  Nest    Source Statement

 271     1    PROCEDURE SALVACP;
 272     1    BEGIN
 273     2        INLINE($E1    /                  (* POP HL *)
 274     2               $22    / XCONTADOR );   (* LD (XCONTADOR),HL *)
 275     2        TABELAP[PCOR].CONTADOR:=XCONTADOR
 276     2    END;
 277     1
 278     1    PROCEDURE TERMEXECUCAO;
 279     1    BEGIN
 280     2        TERMINO:=TRUE
 281     2    END;
 282     1
 283     1
 284     1    PROCEDURE INSPECIONAENTRADA;
 285     1    VAR ESTADOR:BOOLEAN;
 286     2        BUFFER:MSG;
 287     2        TEMPORESPOSTA:INTEGER;
 288     2    BEGIN
 289     2        RECEBE(BUFFER,ESTADOR,TEMPORESPOSTA);
 290     2        IF ESTADOR
 291     2        THEN BEGIN
 292     3             TABELAP[BUFFER[2]].ESTADO:=TRUE;
 293     3             TABELAP[BUFFER[2]].CONTADOR:=BUFFER[3];
 294     3             END;
 295     2    END;
 296     1
 297     1    PROCEDURE SAIDA(CLIENTE:INTEGER;PARAMET:TIPOPAR);
 298     1    VAR BUFFER:MSG;
 299     2        I:INTEGER;
 300     2        ESTADOE:BOOLEAN;
 301     2        CLTA:BYTE;
 302     2    BEGIN
 303     2        CASE CLIENTE OF 1:CLTA:=49;
 304     3                        2:CLTA:=50
 305     3                     END;
 306     2        BUFFER[1]:=RES;
 307     2        FOR I:=1 TO TAMP
 308     2        DO BUFFER[I+3]:=PARAMET[I];
 309     2
 310     2        ENVIA(CLTA,BUFFER,ESTADOE);
 311     2
 312     2        IF NOT ESTADOE
 313     2        THEN WRITE('ERRO:cliente nao recebeu o resultado');
 314     2
 315     2        TABELAP[ID].ESTADO:=FALSE
 316     2    END;
 317     1
 318     1    PROCEDURE CHAMADAREMOTA(SERVIDOR:BYTE;ENDCL:INTEGER;
 319     1              ENDPR:INTEGER;VAR PARAMET:TIPOPAR;
 320     1              VAR ESTADOCRP:INTEGER;TEMPORESPOSTA:INTEGER);
 321     1    VAR RETRANSMISSAO,I:INTEGER;
 322     2        FEITO,ESTADOE,ESTADOR:BOOLEAN;
 323     2        MENSAGEM,RESULT:MSG;
 324     2    BEGIN
```

```
Stmt  Nest    Source Statement

 325     2        MENSAGEM[1]:=CRP;
 326     2        MENSAGEM[2]:=ENDCL;
 327     2        MENSAGEM[3]:=ENDPR;
 328     2        FOR I:=1 TO TAMP
 329     2        DO MENSAGEM[I+3]:=PARAMET[I];
 330     2
 331     2        (* inicializacoes *)
 332     2
 333     2        RETRANSMISSAO:=0;
 334     2        FEITO:=FALSE;
 335     2
 336     2        (* cliente solicita a execucao remota do proced. *)
 337     2
 338     2        REPEAT
 339     3              ENVIA(SERVIDOR,MENSAGEM,ESTADOE);
 340     3              IF ESTADOE
 341     3              THEN FEITO:=TRUE
 342     3              ELSE BEGIN
 343     4                        RETRANSMISSAO:=RETRANSMISSAO + 1;
 344     4                        IF RETRANSMISSAO = MAX
 345     4                        THEN BEGIN
 346     5                                ESTADOCRP:=NAORECEBIDO;
 347     5                                FEITO:=TRUE
 348     5                             END
 349     5                   END
 350     4        UNTIL FEITO;
 351     2
 352     2        (* cliente espera o resultado da CRP *)
 353     2
 354     2        FEITO:=FALSE;
 355     2        IF ESTADOE
 356     2        THEN
 357     2             REPEAT
 358     3                  RECEBE(RESULT,ESTADOR,TEMPORESPOSTA);
 359     3                  IF ESTADOR
 360     3                  THEN IF RESULT[1]= RES
 361     3                       THEN BEGIN
 362     4                                FOR I:=1 TO TAMP
 363     4                                DO PARAMET[I]:=RESULT[I+3];
 364     4                                FEITO:=TRUE;
 365     4                                ESTADOCRP:=EXECUTADO
 366     4                             END
 367     4                  ELSE BEGIN
 368     4                            ESTADOCRP:=NAOEXECUTADO;
 369     4                            FEITO:=TRUE
 370     4                       END
 371     4             UNTIL FEITO;
 372     2    END;
 373     1
 374     1
 375     1
 376     1
 377     1    BEGIN
 378     1        NUMEROSEQUENCIA:=1;
```

```
Stmt  Nest    Source Statement

 379     1        NSEQREC:=1;
 380     1        INICIALIZACAO;
 381     1        SEMPRE:=TRUE;
 382     1
 383     1        WHILE SEMPRE
 384     1        DO BEGIN
 385     2               SELECIONAEXECUCAO;
 386     2               IF TABVAZIA
 387     2               THEN INSPECIONAENTRADA
 388     2               ELSE BEGIN
 389     3                    EXECUCAO;
 390     3                    IF TERMINO
 391     3                    THEN SAIDA(PCOR,PARAMET);
 392     3                    INSPECIONAENTRADA
 393     3                    END
 394     3           END;
 395     1    END.
 395     0    ----------------------------
 395     0    Normal End of Input Reached
```

```
;*****************************************************************
;
;       PROGRAMA QUE FAZ A COMUNICACAO ENTRE  A 8251
;       DO PROCESSADOR OPERADOR E A INTERFACE SIO-Z80
;       DO PROCESSADOR DE COMUNICACAO
;
;       OPERADOR <---> PROCESSADOR DE COMUNICACAO
;
;*****************************************************************
;
;           DESCRICAO DO PROTOCOLO UTILIZADO NA COMUNICACAO
;
;
;           configuracao do formato para o tamanho pequeno
;
;       ---------------------------------------------------
;       ! 2B ! 00 !     DADOS (TAMANHO PEQUENO)     ! D4 ! D4 !
;       ---------------------------------------------------
;
;           configuracao do formato para o tamanho grande
;
;       ---------------------------------------------------
;       ! 2B ! FF !     DADOS (TAMANHO GRANDE)      ! D4 ! D4 '
;       ---------------------------------------------------
;
;
;
;*****************************************************************
;
;       DESCRICAO DA PALAVRA DE ESTADO DA INTERFACE 8251
;
;
;       PALAVRA DE ESTADO
;
;       BIT         FUNCAO
;        0      transmissor ativo
;        1      mensagem completada
;        2      erro de framing
;        3      receptor ativo
;        4      erro de overrun
;        5      estourou o timeout
;        6      recebemos um frame
;        7      erro na recepcao
;
;
;*****************************************************************
;
;       DECLARACAO DAS EQUIVALENCIAS
;
TAMFM   EQU 80          ;TAMANHO DO FRAME
TAMAG   EQU 70          ;TAMANHO FRAME GRANDE
TAMAP   EQU 08          ;TAMANHO FRAME PEQUENO
PORTCTL EQU 20H         ;ENDERECO PORTA CONTROLE 8251
MODO    EQU 20H         ;VALOR PROGRAMACAO
NRTR    EQU 10H         ;CONTROLE DO NRO DE RECEPCOES
```

*** ENTSAIDA ***

```
NRTT    EQU 10H         ;CONTROLE DO NRO DE TRANSMISSOES
RAM     EQU 200H
;
;       DECLARACAO DAS VARIAVEIS
;
        ORG RAM
STCANAL DS 1            ;CONTROLE 8251
TAMGRD  DS 1            ;ARMAZENA TAMANHO DO FRAME GRANDE
TAMPEQ  DS 1            ;ARMAZENA TAMANHO DO FRAME PEQUENO
TIMEOUT DS 2            ;ARMAZENA O TEMPO DE RESPOSTA
INIBR   DS 2            ;ENDERECO INICIO BUFFER RECEPCAO
INIBT   DS 2            ;ENDERECO INICIO BUFFER TRANSMISSAO
TCTRL   DS 1            ;ARMAZENA CAMPO DE CONTROLE
BPED    DS 20           ;BUFFER AUXILIAR DE RECEPCAO E TRANSMISSAO
PIB     DS TAMFM        ;BUFFER DE RECEPCAO
STAT    DS 2            ;ESTADO DA TRANSMISSAO/RECEPCAO
PIBT    DS TAMFM        ;BUFFER DE TRANSMISSAO
NTMS    DS 1            ;CONTROLE DO NRO DE TENTATIVAS DE RECEPCAO
RCBEU   DS 1            ;RECEBEU
TRTIU   DS 1            ;TRANSMITIU
NROT    DS 1            ;CONTROLE DO NRO DE TENTATIVAS DE TRANSMISSAO
CTR     DS 1            ;CONTROLE DO RECEBIMENTO DE RECONHECIMENTO
;
;
;
;       INICIALIZACOES
;
INICIO  LD A,MODO
        OUT (PORTCTL),A
        LD A,0
        LD (CTR),A
        LD  A,1
        LD  (STAT),A
        LD A,TAMAG-4
        LD (TAMGRD),A
        LD A,TAMAP-4
        LD (TAMPEQ),A
        RET
;
;
;
;       RECEBE UM FRAME DO PROCESSADOR DE COMUNICACAO
;
RECEBEF LD A,0
        LD (NTMS),A
        LD (RCBEU),A
RO      CALL ZE8251R
        LD HL,25FFH
        LD (TIMEOUT),HL
        LD HL,BPED
        LD (INIBR),HL
        INC HL
        INC HL
        LD (HL),99H
        CALL RECEBE
```

```
        LD A,(BPED+2)
        CP 0A1H
        JP NZ,CTR0
        CALL PREPBF
        LD HL,BPED
        LD (INIBT),HL
        INC HL
        INC HL
        LD (HL),0B1H
        CALL ZE8251T
        CALL TRANSMIT
        LD HL,PIB
        LD (INIBR),HL
        CALL ZE8251R
        CALL RECEBE
        LD A,(PIB+2)
        LD (TCTRL),A
;
;       TESTA SE TEM ERRO
;
        LD A,(STCANAL)
        BIT 4,A         ;SE ERRO
        JP  NZ,EF2
        BIT 7,A         ;SE ERRO
        JP  NZ,EF2
;
;
;       TESTA O CAMPO DE CONTROLE
;
;            0 - INFORMACAO
;            1 - RECONHECIDO E LIBERADO
;            2 - RECONHECIDO E NAO LIBERADO
;            3 - TRANSMISSAO REJEITADA
;            4 - INICIALIZACAO
;
        LD A,(TCTRL)
        CP 0                    ;TESTA SE O CAMPO CONT.=INFORMACAO
        JP NZ,IC0
        JP ENREC
IC0     CP 1                    ;TESTA SE CAMPO CTRL.=RL
        JP NZ,IC1
        CALL MENRL
        RET
IC1     CP 2                    ;TESTA SE CAMPO CTRL=RNL
        JP NZ,IC2
        CALL MNRNL
        RET
IC2     CP 3                    ;TESTA SE CAMPO CTRL.=TR
        JP NZ,IC3
        CALL TRANN
        RET
IC3     CP 4                    ;TESTA SE CAMPO CTRL.=INICIALIZACAO
        JP NZ,IC4
        CALL INICIO
IC4     RET
```

```
;
;       CONTROLE DO NUMERO DE TENTATIVAS DE RECEPCAO
;
CTRO    LD A,(NTMS)
        INC A
        LD (NTMS),A
        CP NRTR
        JP Z,RETORNO
        JP RO
;
RETORNO LD A,0
        LD (RCBEU),A
        RET
;
;
;       ENREC - ENVIA RECONHECIMENTO
;
ENREC   LD A,1
        LD (CTR),A
        CALL TMRL               ;TRANSMITE - RECEBIDO E LIBERADO
        LD A,1
        LD (RCBEU),A
        RET
;
;       TRATAMENTO DE ERRO
;
EF2     CALL ZE8251R
        LD  A,(STAT)
        CP  3
        JP  Z,RECEBEF
        CALL TMRJT              ;TRANSMITE - TRANSMISSAO REJEITADA
        JP RECEBEF
;
;
;
;        TRANSMITE RECEBIDO E LIBERADO
;
TMRL    LD A,1
        LD (FIBT+2),A
        CALL TRPC
        LD A,1
        LD (STAT),A
        RET
;
;
;       TRANSMITE  - TRANSMISSAO REJEITADA
;
TMRJT   LD A,3
        LD (FIBT+2),A
        CALL TRPC
        RET
;
;
;
;
```

```
;
;         TRANSMITE NOVAMENTE
;
TRANN     CALL TRPC
          RET
;
;         AJUSTA STATUS P/ RECEBIDO E LIBERADO
;
MENRL     LD A,1
          LD (STAT),A
          RET
;
;         AJUSTA STATUS P/ RECEBIDO E NAO LIBERADO
;
MNRNL     LD A,2
          LD (STAT),A
          RET
;
;        TRANSMITE UM FRAME PARA O PROCESSADOR DE COMUNICACAO
;
;
TRPC      LD A,0
          LD (NROT),A
          LD (TRTIU),A
TRPRO     CALL ZE8251T
          LD HL,05FFH
          LD (TIMEOUT),HL
          CALL PREPBF
          LD HL,BPED
          LD (INIBT),HL
          INC HL
          INC HL
          LD (HL),0A1H
          CALL TRANSMIT
          LD HL,BPED
          LD (INIBR),HL
          CALL ZE8251R
          INC HL
          INC HL
          LD (HL),99H
          CALL RECEBE
          LD A,(BPED+2)
          CP 0B1H
          JP NZ,CTR1
          LD HL,PIBT
          LD (INIBT),HL
          PUSH HL
          POP IX
          LD (IX),02BH
          LD (IX+1),0FFH
          LD (IX+TAMFM-4),0D4H
          LD (IX+TAMFM-3),0D4H
          CALL ZE8251T
          CALL TRANSMIT
          LD A,3
```

*** ENTSAIDA ***

```
        LD (STAT),A
        LD A,(CTR)
        CP 1
        JP Z,TRP0
        CALL RECEBEF
        LD A,1
        LD (TRTIU),A
TRP0    LD A,0
        LD (CTR),A
        RET
;
;       CONTROLE DO NRO DE TENTATIVAS DE TRANSMISSAO
;
CTR1    LD A,(NROT)
        INC A
        LD (NROT),A
        CP NRTT
        JP Z,RETOR1
        JP TRPRO
RETOR1  LD A,0
        LD (TRTIU),A
        RET
;
;
;       PREPARA O BUFFER AUXILIAR PARA FAZER UMA TRANSMISSAO
;
PREPBF  LD IX,BPED
        LD (IX),02BH
        LD (IX+1),0H
        LD (IX+6),0D4H
        LD (IX+7),0D4H
        RET
;
;
;       INICIALIZA A PALAVRA DE CONTROLE DA 8251
;       PARA A RECEPCAO
;
ZE8251R LD HL,STCANAL
        RES 3,(HL)
        RES 4,(HL)
        RES 5,(HL)
        RES 6,(HL)
        RES 7,(HL)
        RET
;
;
;       INICIALIZA A PALAVRA DE CONTROLE DA 8251
;       PARA A TRANSMISSAO
;
ZE8251T LD HL,STCANAL
        RES 0,(HL)
        RES 1,(HL)
        RES 2,(HL)
        RET
;
```

```
;
;
;*****************************************************************************
;
;       ROTINA DE RECEPCAO DE UMA CADEIA DE BYTES
;
;       CASO A RECEPCAO DO FRAME SE INTERROMPA, O COMANDO E
;       DEVOLVIDO PARA O PROGRAMA PRINCIPAL.
;
;       SEMPRE QUE UM CARACTERE DEMORA MAIS DO QUE O  TEMPO
;       DO TIMEOUT PARA CHEGAR, O COMANDO E DEVOLVIDO AO
;       PROGRAMA PRINCIPAL, E E ASSINALADA A CONDICAO DE
;       ESTOURO DO TEMPO DE RESPOSTA.
;
;*****************************************************************************
RECEBE   RET
;
;
;
;*****************************************************************************
;
;       ROTINA DE TRANSMISSAO DE UMA CADEIA DE BYTES
;
;       TRANSMITE UMA CADEIA DE BYTES E RETORNA
;       PARA O PROGRAMA PRINCIPAL
;
;*****************************************************************************
TRANSMIT RET
;
;
        END
```

A P Ê N D I C E A 2

LISTAGEM DO SUPERVISOR DO PROCESSADOR DE COMUNICAÇÃO

```
;         PROGRAMA SUPERVISOR DO PROCESSADOR DE COMUNICACAO
;         PROC. DE COMUNICACAO <-----> PROC. DE COMUNICACAO
;         PROCESSADOR OPERADOR <-----> PROC. DE COMUNICACAO
;*****************************************************************************
;
;        ESPECIFICACAO DA PALAVRA DE  'STATUS' DO SIO (SIOAST ou SIOBST)
;
;        BIT          FUNCAO
;         0     transmissor ativo
;         1     mensagem completada
;         2     transmissao abortada
;         3     receptor ativo
;         4     erro de overrun
;         5     erro de crc
;         6     fim de frame
;         7     detectado aborto na recepcao
;
;*****************************************************************************
;
;           EQUIVALENCIAS GERAIS DO PROGRAMA
;
RAM       EQU   1000H       ;
EPROM     EQU   0000H       ;
NFRAME    EQU   05          ;NRO DE FRAMES POR FILA
TAMFM     EQU   80          ;TAMANHO DO FRAME
TAMAG     EQU   70          ;TAMANHO DO FRAME GRANDE
TAMAP     EQU   08          ;TAMANHO DO FRAME PEQUENO
;
          ORG RAM
SIOAST   DS 1               ;CONTROLE SIO PORTA A
ST8251   DS 1               ;CONTROLE 8251
TAMGRD   DS 1               ;ARMAZENA TAMANHO DO FRAME GRANDE
TAMPEQ   DS 1               ;ARMAZENA TAMANHO DO FRAME PEQUENO
TIMEOUT  DS 2               ;ARMAZENA O TEMPO DE RESPOSTA
INIBT8   DS 2               ;ENDERECO INICIO BUFFER TRANSM. 8251
INIBR8   DS 2               ;ENDERECO INICIO BUFFER TRANSM. 8251
NBYTEA   DS 2               ;NRO BYTES TRANSMITIR SIO-A
INIBTA   DS 2               ;ENDERECO INICIO BUFFER TRANSM. SIO-A
INIBRA   DS 2               ;ENDERECO INICIO BUFFER REC. SIO-A
SIOBST   DS 1               ;CONTROLE SIO PORTA B
NBYTEB   DS 2               ;NRO BYTES TRANSMITIR SIO-B
INIBTB   DS 2               ;ENDERECO INICIO BUFFER TRANSM. SIO-B
INIBRB   DS 2               ;ENDERECO INICIO BUFFER REC. SIO-B
P        DS 1               ;NUMERO DO BUFFER ACESSADO
CAM      DS 1               ;CAMINHO QUE O FRAME DEVE SEGUIR
TESPL    DS 2               ;TESTA ESPACO LIVRE DA FILA
TCTRL    DS 1               ;ARMAZENA CAMPO DE CONTROLE
TSTAT    DS 1               ;ARMAZENA O STATUS
TPIB     DS 2               ;TESTA O BUFFER
TPIBR    DS 2
STAT1    DS 1               ;STATUS 1
STAT2    DS 1               ;STATUS 2
STAT3    DS 1               ;STATUS 3
TOUT1    DS 2               ;TIME-OUT 1
TOUT2    DS 2               ;TIME-OUT 2
```

```
TOUT3   DS 2                    ;TIME-OUT 3
TC      DS 1                    ;TRANSMISSAO CORRETA
FNC     DS 1                    ;FILA NAO CHEIA
NZV     DS 2                    ;CONTROLE DE RETRANSMISSAO
TLIMIT  DS 2                    ;TIME-OUT LIMITE
NBT     DS 1                    ;NRO DO BUFFER TRANSMITIR
TOL3    DS 2                    ;TIME-OUT LIMITE DA 8251
TSIO    DS 2                    ;
TTO     DS 2                    ;
BFED    DS 20                   ; BUFFER AUXILIAR DA 8251
AUX     DS 20                   ; BUFFER AUXILIAR
;
ESPL1   DS 2                    ;--------------------
ESPL2   DS 2                    ;- ESPACO LIVRE DA FILA(I)
ESPL3   DS 2                    ;--------------------
;
PCF1    DS 2                    ;--------------------
PCF2    DS 2                    ;- PONTEIRO COMECO DA FILA(I)
PCF3    DS 2                    ;--------------------
;
PFFL1   DS 2                    ;--------------------
PFFL2   DS 2                    ;- PONTEIRO FIM DA FILA(I)
PFFL3   DS 2                    ;--------------------
;
PIBT1   DS TAMFM                ;--------------------
PIBT2   DS TAMFM                ;- PONTEIRO INICIO BUFFER TRANSMISSAO(I)
PIBT3   DS TAMFM                ;--------------------
;
PIB1    DS TAMFM                ;--------------------
PIB2    DS TAMFM                ;- PONTEIRO INICIO DO BUFFER DE RECEPCAO(I)
PIB3    DS TAMFM                ;--------------------
;
PIF1    DS NFRAME*TAMFM         ;--------------------
PIF2    DS NFRAME*TAMFM         ;-  INICIO POSICOES RESERVADAS FILA(I)
PIF3    DS NFRAME*TAMFM         ;--------------------
PIF4    EQU $
;
PFF1    EQU PIF2                ;--------------------
PFF2    EQU PIF3                ;- FIM POSICOES RESERVADAS FILA(I)
PFF3    EQU PIF4                ;--------------------
;
TAMANHO EQU 300                 ;
PILHA   EQU $                   ;
;
        ORG  EPROM
INICIO  LD SP,PILHA
        LD HL,PIF1              ;--------------------
        LD (PCF1),HL            ;- INICIALIZACOES DAS FILAS
        LD (PFFL1),HL           ;-
        LD  HL,PIF2             ;- PONTEIRO COMECO DA FILA(I) RECEBE
        LD (PCF2),HL            ;- INICIO POSICOES RESERVADAS PARA FILA(I)
        LD (PFFL2),HL           ;-
        LD  HL,PIF3             ;-
        LD (PCF3),HL            ;-
        LD (PFFL3),HL           ;--------------------
```

```
        LD HL,NFRAME*TAMFM  ;--------------------
        LD  (ESPL1),HL      ;- INICIALIZACAO DO ESPACO LIVRE NA FILA(I)
        LD  (ESPL2),HL      ;-
        LD  (ESPL3),HL      ;-----------------------
;
        LD  A,0             ;--------------------
        LD  (P),A           ;- INICIALIZA VARIAVEIS COM ZERO
        LD  (NBT),A         ;-
        LD  (TC),A          ;-
        LD  (FNC),A         ;-
        LD  (TOUT1),A       ;-
        LD  (TOUT2),A       ;-
        LD  (TOUT3),A       ;----------------------
;
        LD  A,1             ;--------------------
        LD  (STAT1),A       ;- INICIALIZA STATUS COM RECEBIDO E LIBERADO
        LD  (STAT2),A       ;-
        LD  (STAT3),A       ;---------------------
;
        LD  A,15H           ;--------------------
        LD  (TLIMIT),A      ;- INICIALIZA O TEMPO MAXIMO DE ESPERA POR
        LD  A,6             ;- UMA TRANSMISSAO
        LD  (TOL3),A        ;- TEMPO MAXIMO DE ESPERA DA 8251
        LD  A,0FFH          ;-
        LD  (NZV),A         ;- E O NUMERO DE RETRANSMISSOES
                            ;--------------------
;
        LD  HL,PIBT1        ;--------------------
        LD  (INIBTA),HL     ;- INICIALIZA ENDERECO DO BUFFER DE
        LD  HL,PIB1         ;- TRANSMISSAO E RECEPCAO
        LD  (INIBRA),HL     ;-
        LD  HL,PIBT2        ;-
        LD  (INIBTB),HL     ;-
        LD  HL,PIB2         ;-
        LD  (INIBRB),HL     ;--------------------
;
        LD A,TAMAG-4
        LD (TAMGRD),A
        LD A,TAMAP-4
        LD (TAMPEQ),A
;
;       VERIFICA AS RECEPCOES
;
R0      LD A,(P)
        INC  A
        LD   (P),A
        CP 4                ;TESTA SE JA PERCORREU OS 3 BUFFERS
        JP Z,ROTO
        CP 1                ;TESTA SE E O BUFFER 1
        JP NZ,R1
        LD BC,SIOAST
        LD A,(BC)
        BIT 6,A
        JP Z,RF
        LD (TSIO),BC        ;TRANSFERE SIOST PARA TSIO
```

```
        LD HL,(ESPL1)
        LD (TESPL),HL       ;TRANSFERE ESPACO LIVRE 1 PARA TESPL
        LD DE,PIB1
        LD (TPIBR),DE       ;
        LD DE,PIBT1
        LD (TPIB),DE        ;TRANSFERE ENDERECO BUFFER DE TRANSMISSAO
                            ;PARA TPIB
        LD A,(PIB1)
        LD (TCTRL),A        ;TRANSFERE CAMPO DE CONTROLE PARA TCTRL
        LD A,(STAT1)
        LD (TSTAT),A        ;TRANSFERE O STATUS1 PARA TSTAT
        LD HL,TOUT1
        LD (TTO),HL         ;TRANSFERE O TIME OUT 1 PARA TTO
        JP LA
;
R1      CP 2                ;TESTA SE E O BUFFER 2
        JP NZ,R2
        LD BC,SIOBST
        LD A,(BC)
        BIT 6,A
        JP Z,RF
        LD (TSIO),BC
        LD HL,(ESPL2)
        LD (TESPL),HL
        LD DE,PIB2
        LD (TPIBR),DE
        LD DE,PIBT2         ;
        LD (TPIB),DE
        LD A,(PIB2)
        LD (TCTRL),A
        LD A,(STAT2)
        LD (TSTAT),A
        LD HL,TOUT2
        LD (TTO),HL
        JP LA
;
;       RECEPCAO DO OPERADOR
;
R2      LD HL,05FFH
        LD (TIMEOUT),HL
        LD HL,BPED          ;
        LD (INIBR8),HL
        INC HL
        INC HL
        LD (HL),99H
        CALL ZE8251R
        CALL RECEBE
        LD A,(BPED+2)
        CP 0A1H
        JP NZ,R0
        CALL PREPBF
        LD HL,BPED
        LD (INIBT8),HL
        INC HL
        INC HL
```

```
        LD (HL),0B1H
        CALL ZE8251T
        CALL TRANSMIT
        LD HL,PIB3
        LD (INIBR8),HL
        CALL ZE8251R
        CALL RECEBE
        LD BC,ST8251
        LD (TSIO),BC
        LD HL,(ESPL3)
        LD (TESPL),HL
        LD DE,PIB3+2
        LD (TPIBR),DE
        LD DE,PIBT3+2
        LD (TPIB),DE
        LD A,(PIB3+2)
        LD (TCTRL),A
        LD A,(STAT3)
        LD (TSTAT),A
        LD HL,TOUT3
        LD (TTO),HL
;
;
;       TESTA SE TEM ERRO
;
LA      LD BC,(TSIO)
        LD A,(BC)
        BIT 4,A
        JP  NZ,EF2
        BIT 5,A
        JP  NZ,EF2
        BIT 7,A
        JP  NZ,EF2
;
;
        LD HL,(TSIO)
        RES   6,(HL)
;
;
;
;        TESTA O CAMPO DE CONTROLE
;
;            0 - INFORMACAO
;            1 - RECONHECIDO E LIBERADO
;            2 - RECONHECIDO E NAO LIBERADO
;            3 - TRANSMISSAO REJEITADA
;            4 - INICIALIZACAO
;
        LD A,(TCTRL)
        CP 0                    ;TESTA SE O CAMPO CONT.=INFORMACAO
        JP NZ,IC0
        JP EFER
IC0     CP 1                    ;TESTA SE CAMPO CTRL.=RL
        JP NZ,IC1
        CALL MENRL
```

```
        JP RO
IC1     CP 2                    ;TESTA SE CAMPO CTRL=RNL
        JP NZ,IC2
        CALL MNRNL
        JP RO
IC2     CP 3                    ;TESTA SE CAMPO CTRL.=TR
        JP NZ,IC3
        CALL TRANN
        JP RO
IC3     CP 4                    ;TESTA SE CAMPO CTRL.=INICIALIZACAO
        JP NZ,IC4
        JP INICIO
IC4     JP RO
;
;   EFER - ENCAMINHA FILA E ENVIA RECONHECIMENTO
;
;       TC=0 - TRANSMISSAO CORRETA
;       TC=1 - TRANSMISSAO INCORRETA
;       FNC=0 - FILA NAO CHEIA
;       FNC=1 - FILA CHEIA
;
EFER    CALL ROTA
        LD A,(CAM)
        CP 1                     ;TESTA SE E PARA A FILA 1
        JP NZ,R00
        CALL EXEC1
TFC     LD A,(FNC)
        CP 0                     ;TESTA SE FILA NAO CHEIA
        JP NZ,EF1
        CALL TMRL                ;TRANSMITE - RECEBIDO E LIBERADO
        JP RO
EF1     CALL TMRNL               ;TRANSMITE - RECEBIDO E NAO LIBERADO
        JP RO
;
;
;       TRATAMENTO DE ERRO
;
EF2     LD HL,(TSIO)
        RES 3,(HL)
        RES 4,(HL)
        RES 5,(HL)
        RES 6,(HL)
        RES 7,(HL)
        LD  A,(TSTAT)
        CP  3
        JP  Z,RO
        CALL TMRJT               ;TRANSMITE - TRANSMISSAO REJEITADA
        JP RO
;
;       TESTA SE E PARA FILA 2
;
R00     CP 2
        JP NZ,R11
        CALL EXEC2
        JP TFC
```

```
;
;       TESTA SE E PARA FILA 3
;
R11     CP 3
        JP NZ,R0
        CALL EXEC3
        JP TFC
;
;       VERIFICA SE A FILA 1 ESTA NO FIM DAS SUAS POSICOES
;
EXEC1   LD  HL,PFF1
        LD IY,PFFL1
        LD A,(IY)
        CP L
        JP NZ,CC1
        LD A,(IY+1)
        CP H
        JP Z,R000
CC1     CALL EXEC11
        RET
;
;       VERIFICA SE A FILA 2 ESTA NO FIM DAS SUAS POSICOES
;
;
EXEC2   LD  HL,PFF2
        LD IY,PFFL2
        LD A,(IY)
        CP L
        JP NZ,CC2
        LD A,(IY+1)
        CP H
        JP Z,R111
CC2     CALL EXEC22
        RET
;
;       VERIFICA SE A FILA 3 ESTA NO FIM DAS SUAS POSICOES
;
EXEC3   LD HL,PFF3
        LD IY,PFFL3
        LD A,(IY)
        CP L
        JP NZ,CC3
        LD A,(IY+1)
        CP H
        JP Z,R222
CC3     CALL EXEC33
        RET
;
;       COLOCA A MENSAGEM RECEBIDA NA FILA 1
;
EXEC11  LD HL,(TPIBR)
        LD DE,(PFFL1)
        LD BC,TAMFM
        LDIR
        LD (PFFL1),DE
```

```
        LD    HL,(ESPL1)
        LD    BC,TAMFM
        SCF
        CCF
        SBC   HL,BC
        LD    (ESPL1),HL
        LD    A,H
        CP 0
        JP NZ,EXF1
        LD  A,L
        CP 0
        JP  NZ,EXF1
        LD  A,1
        LD  (FNC),A
EXF1    RET
;
;       COLOCA A MENSAGEM RECEBIDA NA FILA 2
;
EXEC22  LD HL,(TPIBR)
        LD DE,(PFFL2)
        LD BC,TAMFM
        LDIR
        LD (PFFL2),DE
        LD    HL,(ESPL2)
        LD    BC,TAMFM
        SCF
        CCF
        SBC   HL,BC
        LD    (ESPL2),HL
        LD    A,H
        CP    0
        JP    NZ,EXF2
        LD    A,L
        CP    0
        JP    NZ,EXF2
        LD    A,1
        LD    (FNC),A
EXF2    RET
;
;       COLOCA A MENSAGEM RECEBIDA NA FILA 3
;
EXEC33  LD HL,(TPIBR)
        LD DE,(PFFL3)
        LD BC,TAMFM
        LDIR
        LD (PFFL3),DE
        LD HL,(ESPL3)
        LD BC,TAMFM
        SCF
        CCF
        SBC HL,BC
        LD (ESPL3),HL
        LD A,H
        CP 0
        JP NZ,EXF3
```

```
        LD A,L
        CP 0
        JP NZ,EXF3
        LD A,1
        LD (FNC),A
EXF3    RET
;
;       DESLOCAR AS MENSAGENS DA FILA 1 PARA RECUPERAR
;       ESPACO DE ARMAZENAMENTO
;
R000    LD  HL,(PCF1)
        LD IX,PFFL1
        LD  A,H
        CP  (IX+1)
        JP  NZ,RD0
        LD  A,L
        CP   (IX)
        JP  NZ,RD0
        LD  HL,PIF1
        LD  (PCF1),HL
        LD  (PFFL1),HL
        CALL EXEC11
        RET
RD0     LD DE,PIF1
        LD   HL,NFRAME*TAMFM
        LD   BC,(ESPL1)
        SCF
        CCF
        SBC HL,BC
        LD  B,H
        LD  C,L
        LDIR
        LD IX,PIF1
        LD (PCF1),IX
        LD (PFFL1),DE
        CALL EXEC11
        RET
;
;       DESLOCAR AS MENSAGENS DA FILA 2 PARA RECUPERAR
;       ESPACO DE ARMAZENAMENTO
;
;
R111    LD  HL,(PCF2)
        LD IX,PFFL2
        LD  A,H
        CP (IX+1)
        JP  NZ,RD1
        LD  A,L
        CP  (IX)
        JP  NZ,RD1
        LD  HL,PIF2
        LD  (PCF2),HL
        LD  (PFFL2),HL
        CALL EXEC22
        RET
```

```
RD1     LD  DE,PIF2
        LD   HL,NFRAME*TAMFM
        LD   BC,(ESPL2)
        SCF
        CCF
        SBC  HL,BC
        LD   B,H
        LD   C,L
        LDIR
        LD IX,PIF2
        LD (PCF2),IX
        LD (PFFL2),DE
        CALL EXEC22
        RET
;
;       DESLOCAR AS MENSAGENS DA FILA 3 PARA RECUPERAR
;       ESPACO DE ARMAZENAMENTO
;
R222    LD HL,(PCF3)
        LD IX,PFFL3
        LD A,H
        CP (IX+1)
        JP NZ,RD2
        LD A,L
        CP (IX)
        JP NZ,RD2
        LD HL,PIF3
        LD (PCF3),HL
        LD(PFFL3),HL
        CALL EXEC33
        RET
RD2     LD DE,PIF3
        LD HL,NFRAME*TAMFM
        LD BC,(ESPL3)
        SCF
        CCF
        SBC HL,BC
        LD B,H
        LD C,L
        LDIR
        LD IX,PIF3
        LD (PCF3),IX
        LD (PFFL3),DE
        CALL EXEC33
        RET
;
;
;
;
;
;
;
;
;
;
```

```
;*****************************************************************************
;
;       ROTINA QUE DETERMINA O CAMINHO QUE A MEMSAGEM DEVE
;       SEGUIR.
;
;       VARIAVEL CAM RECEBE O NRO DA FILA PARA ONDE A
;       MENSAGEM DEVE SER COLOCADA.
;
;*****************************************************************************
ROTA    RET
;*****************************************************************************



;
;
;
;
;       TESTA O STATUS E O TIME-OUT
;
;          STATUS =  3 - ESPERANDO CONFIRMACAO
;                    2 - RECEBIDO E NAO LIBERADO
;                    1 - RECEBIDO E LIBERADO
;
RF      LD A,(TSTAT)
        CP 3                        ;TESTA SE STATUS=ESPERANDO CONFIRMACAO
        JP NZ,TRNL
        LD HL,(TTO)
        LD A,(HL)
        PUSH HL
        LD B,A
        LD A,(P)
        CP 3
        JP Z,RF01
        LD HL,TLIMIT
RF00    LD A,B
        CP (HL)                 ;TESTA SE O TIME-OUT ULTRAPASSOU O TEMPO
                                ;LIMITE
        POP HL
        JP Z,NVEZ
        INC A
        LD  (HL),A
        JP RO
RF01    LD HL,TOL3
        JP RF00
NVEZ    LD  (HL),0
        LD A,(NZV)
        DEC A
        LD  (NZV),A
        CP 0                        ;TESTA SE E N-EZIMA VEZ
        JP Z,ER2
;
;
        LD BC,(TSIO)
        LD A,(BC)
```

```
        BIT 1,A
        JP  Z,TNC01
        BIT 2,A
        JP  Z,TCORT
TNC01   CALL TRANN
        JP  R0
TCORT   CALL TORJT
ER2     JP R0
TRNL    CP 2                    ;TESTA SE STATUS=RNL
        JP NZ,R0
        PUSH HL
        LD HL,(TESPL)
        LD A,H
        CP 0
        JP NZ,TRC01
        LD A,L
        CP 0
        JP NZ,TRC01
        POP HL
        JP R0
TRC01   POP HL
        LD  A,1
        LD  (TSTAT),A
        LD  A,(P)
        CALL AJST
        CALL TMRL
        JP R0
;
;
;     TRANSMITE RECEBIDO E LIBERADO
;
TMRL    LD A,1
        LD   HL,(TPIB)
        LD   (HL),A
        LD  A,(P)
        CP  1
        JP  NZ,TMRL1
        CALL TRPRA
        JP  TMRL3
TMRL1   CP 2
        JP NZ,TMRL2
        CALL TRPRB
        JP  TMRL3
TMRL2   CALL TRPRO
        LD A,1
        LD (TSTAT),A
        LD A,(P)
        CALL AJST
TMRL3   RET
;
;
;     TRANSMITE RECEBIDO E NAO LIBERADO
;
TMRNL   LD A,2
        LD    HL,(TPIB)
```

```
          LD    (HL),A
          LD  A,(P)
          CP 1
          JP NZ,TMRNL1
          CALL TRPRA
          JP  TMRNL3
TMRNL1    CP 2
          JP NZ,TMRNL2
          CALL TRPRB
          JP TMRNL3
TMRNL2    CALL TRPRO
          LD A,2
          LD (TSTAT),A
          LD A,(P)
          CALL AJST
TMRNL3    RET
;       TRANSMITE  - TRANSMISSAO REJEITADA
;
TMRJT     LD A,3
          LD   HL,(TPIB)
          LD   (HL),A
          LD A,(P)
          CP 1
          JP  NZ,TMRJT1
          CALL TRPRA
          JP TMRJT3
TMRJT1     CP 2
          JP NZ,TMRJT2
          CALL TRPRB
          JP TMRJT3
TMRJT2    CALL TRPRO
TMRJT3    RET
;
;      TRANSMITE NOVAMENTE
;
TRANN     LD A,(P)
          CP 1
          JP NZ,TRANN1
          CALL TRPRA
          JP TRANN3
TRANN1    CP 2
          JP NZ,TRANN2
          CALL TRPRB
          JP TRANN3
TRANN2    CALL TRPRO
TRANN3    LD HL,(TTO)
          LD (HL),1
          RET
;
;      AJUSTA STATUS P/ RECEBIDO E LIBERADO
;
MENRL     LD A,1
          LD (TSTAT),A
          LD  A,(P)
          CALL AJST
```

```
          LD HL,(TTO)
          LD (HL),0
          RET
;
;      AJUSTA STATUS P/ RECEBIDO E NAO LIBERADO
;
MNRNL     LD A,2
          LD (TSTAT),A
          LD A,(P)
          CALL AJST
          LD HL,(TTO)
          LD (HL),0
          RET
;
;
;         AJUSTA O STATUS
;
AJST      CP 1
          JP NZ,AJ1
          LD A,(TSTAT)
          LD (STAT1),A
          RET
AJ1       CP 2
          JP NZ,AJ2
          LD A,(TSTAT)
          LD (STAT2),A
AJ2       CP 3
          JP NZ,AJ3
          LD A,(TSTAT)
          LD (STAT3),A
AJ3       RET
;
;
;
;          VERIFICA AS TRANSMISSOES
;
;
ROTO      LD  A,0
          LD  (P),A
          LD  A,(NBT)
          INC A
          LD  (NBT),A
          CP  4
          JP  Z,SF
          CP  1          ;TESTA SE O PROXIMO BUFFER DE TRANSMISSAO E 1
          JP  NZ,ROT1
ROTOO     LD   HL,(ESPL1)
          LD   BC,NFRAME*TAMFM
          LD   A,H
          CP   B
          JP   NZ,ROTE1
          LD   A,L
          CP   C
          JP   NZ,ROTE1
          JP   ROT5
```

```
ROTE1     LD A,(STAT1)
          CP 1                ;TESTA SE A TRANSMISSAO ESTA LIBERADA
          JP NZ,ROT5
          PUSH DE
          PUSH HL
          CALL DELET1
          POP HL
          POP DE
          JP ROT4
;
;         TESTA SE O PROXIMO BUFFER DE TRANSMISSAO E 2
;
ROT1      CP 2
          JP NZ,ROT2
          LD   HL,(ESPL2)
          LD   BC,NFRAME*TAMFM
          LD   A,H
          CP B
          JP   NZ,ROTE2
          LD   A,L
          CP C
          JP   NZ,ROTE2
          JP   ROT5
ROTE2     LD A,(STAT2)
          CP 1              ;TESTA SE A TRANSMISSAO ESTA LIBERADA
          JP NZ,ROT5
          PUSH DE
          PUSH HL
          CALL DELET2
          POP HL
          POP DE
          JP ROT4
;
;         TESTA SE O PROXIMO BUFFER DE TRANSMISSAO E 3
;
ROT2      CP 3
          JP NZ,RO
          LD HL,(ESPL3)
          LD BC,NFRAME*TAMFM
          LD A,H
          CP B
          JP NZ,ROTE3
          LD A,L
          CP C
          JP NZ,ROTE3
          JP ROT5
ROTE3     LD A,(STAT3)
          CP 1
          JP NZ,ROT5
          PUSH DE
          PUSH HL
          CALL DELET3
          POP HL
          POP DE
          CALL TRPRO
```

```
        JP R0
;
;
ROT4    LD HL,SIOAST
        RES 0,(HL)
        RES 1,(HL)
        RES 2,(HL)
        LD  HL,TAMFM-2
        LD  (NBYTEA),HL
        CALL TRANSA
        LD A,3
        LD (TSTAT),A
        LD A,(NBT)
        CALL AJST
        JP   R0
;
;
ROT5    LD   A,(NBT)
        CP   3
        JP   Z,R0
        JP   ROT0
;
;       RETIRA MENSAGEM DA FILA 1
;
DELET1  LD HL,(PCF1)
        LD DE,PIBT1
        LD BC,TAMFM
        LDIR
        LD (PCF1),HL
        LD  HL,(ESPL1)
        LD  BC,TAMFM
        SCF
        CCF
        ADC  HL,BC
        LD   (ESPL1),HL
        RET
;
;       RETIRA MENSAGEM DA FILA 2
;
DELET2  LD HL,(PCF2)
        LD DE,PIBT2
        LD BC,TAMFM
        LDIR
        LD (PCF2),HL
        LD HL,(ESPL2)
        LD BC,TAMFM
        SCF
        CCF
        ADC HL,BC
        LD  (ESPL2),HL
        RET
;
;       RETIRA MENSAGEM DA FILA 3
;
DELET3  LD HL,(PCF3)
```

```
        LD DE,PIBT3+2
        LD BC,TAMFM-2
        LDIR
        INC HL
        INC HL
        LD (PCF3),HL
        LD HL,(ESPL3)
        LD BC,TAMFM
        SCF
        CCF
        ADC HL,BC
        LD (ESPL3),HL
        RET
;
;
;
SF      LD  A,1
        LD  (NBT),A
        JP ROTOO
;
;
;
;
;       TRANSMITE PELA PORTA A DA SIO
;
TRPRA   LD HL,TAMFM-2
        LD (NBYTEA),HL
        LD HL,SIOAST
        RES 0,(HL)
        RES 1,(HL)
        RES 2,(HL)
        CALL TRANSA
        RET
;
;       TRANSMITE PELA PORTA B DA SIO
;
TRPRB   LD HL,TAMFM-2
        LD (NBYTEB),HL
        LD HL,SIOBST
        RES 0,(HL)
        RES 1,(HL)
        RES 2,(HL)
        CALL TRANSB
        RET
;
;       TRANSMITE PARA O OPERADOR
;
TRPRO   CALL ZE8251T
        LD HL,03FFH
        LD (TIMEOUT),HL
        CALL PREPBF
        LD HL,BPED
        LD (INIBT8),HL
        INC HL
        INC HL
```

```
        LD (HL),0A1H
        CALL TRANSMIT
        LD HL,BPED
        LD (INIBR8),HL
        CALL ZE8251R
        INC HL
        INC HL
        LD (HL),99H
        CALL RECEBE
        LD A,(BPED+2)
        CP 0B1H
        JP NZ,TRPRO
        LD HL,PIBT3
       LD (INIBT8),HL
        PUSH HL
        POP IX
        LD (IX),02BH
        LD (IX+1),0FFH
        LD (IX+TAMFM-4),0D4H
        LD (IX+TAMFM-3),0D4H
        CALL ZE8251T
        CALL TRANSMIT
        LD A,(P)
        CP 0
        JP NZ,FORAT
        LD A,3
        LD (TSTAT),A
        LD A,(NBT)
        CALL AJST
FORAT   RET
;
;       TRANSMITE TRANSMISSAO REJEITADA
;
TORJT   PUSH HL
        LD   A,3
        LD   (AUX),A
        LD   HL,4
        LD   (NBYTEA),HL
        LD   HL,AUX
        LD   (INIBTA),HL
        LD   A,0
        LD   HL,SIOAST
        LD   (HL),A
        CALL TRANSA
        POP  HL
        RET
;
;       PREPARA O BUFFER DE TRANSMISSAO
;
PREPBF  LD IX,BPED
        LD (IX),02BH
        LD (IX+1),0H
        LD (IX+6),0D4H
        LD (IX+7),0D4H
        RET
```

```
;
;       INICIALIZA A PALAVRA DE CONTROLE DA 8251
;       PARA A RECEPCAO
;
ZE8251R LD HL,ST8251
        RES 3,(HL)
        RES 4,(HL)
        RES 5,(HL)
        RES 6,(HL)
        RES 7,(HL)
        RET
;
;       INICIALIZA A PALAVRA DE CONTROLE DA 8251
;       PARA A TRANSMISSAO
;
ZE8251T LD HL,ST8251
        RES 0,(HL)
        RES 1,(HL)
        RES 2,(HL)
        RET
;
;
;
TRANSB  RET
;*****************************************************************************
;
;     ROTINA DE TRANSMISSAO DE UMA CADEIA DE BYTES PELA PORTA B DA
;     INTERFACE SERIAL SIO-Z80
;
;*****************************************************************************
;
;
TRANSA  RET
;*****************************************************************************
;
;       ROTINA DE TRANSMISSAO DE UMA CADEIA DE BYTES PELA PORTA A DA
;       INTERFACE SERIAL SIO-Z80
;
;*****************************************************************************
;
;
TRANSMIT RET
;*****************************************************************************
;
;       ROTINA DE TRANSMISSAO DE UMA CADEIA DE BYTES PELA INTERFACE
;       SERIAL SIO-Z80 PARA O PROCESSADOR OPERADOR
;
;*****************************************************************************
;
;
;
;
;
;
;
```

```
;
RECEBE   RET
;*****************************************************************
;
;       ROTINA DE RECEPCAO DE UMA CADEIA DE BYTES PELA INTERFACE SERIAL
;       SIO-Z80 DO PROCESSADOR OPERADOR
;
;*****************************************************************
;
;
;
         END
```

REFERÊNCIAS BIBLIOGRÁFICAS

|1| DIJKSTRA, E. W. - "The Structure of the "THE" Multiprogramming System" - Commun. ACM, 11,5, (May 1968), pg 341-346.

|2| JONES, A. K., e SCHWARZ, P. - "Experience Using Multiprocessor Systems - A Status Report" - ACM Comput. Surv.,12,2, (June 1980), pg 121-165.

|3| DIJKSTRA, E. W. - "Solution of a Problem in Concurrent Programming Control". - Commun. ACM, 8,9, (Sept. 1965), pg 569.

|4| DIJKSTRA, E. W. - "Cooperating Sequential Processes" em Programming Languages, (ed. F. Genuys), Academic Press, NY, 1968, pg 43-112.

|5| KNUTH, D. E. - "Additional Comments on a Problem in Concurrent Programming Control" - Commun. ACM, 9,5, (May 1966),

pg 321-322.

|6| BRUIJN, N. G. - "Additional Comments on a Problem in Concurrent Programming Control" - Commun. ACM, 10,3, (Mar 1967), pg 137-138.

|7| EISENBERG, M. A. e McGUIRE, M. R. - "Further Comments on Dijkstra's Concurrent Programming Control Problem" - Commun. ACM, 15,11, (Nov 1972), pg 999.

|8| LAMPORT, L. - " A New Solution to Dijkstra's Concurrent Programming Problem" - Commun. ACM, 14,8, (Aug 1974), pg 453-455.

|9| HOARE, C. A. R. - "Towards a Theory of Parallel Programming" em Operating Systems Techniques, Academic Press, NY, 1972, pg 61-71.

|10| BRINCH HANSEN, P. - "Operating Systems Principles", Prentice-Hall, NJ, 1973.

|11| HOARE, C. A. R. - "Monitors: An Operating System Structuring Concept" - Commun. ACM, 17, 10, (Oct 1974), pg 61-71.

|12| BRINCH HANSEN, P. - "The Programming Language Concurrent

Pascal", - IEEE Trans. Soft. Eng., 1, 2, (June 1975), pg 199-207.

|13| WIRTH, N. - "Modula: A Language for Modular Multiprogramming" - Software: Practice and Experience ,7 , 1, (Jan 1977), pg 3-35.

|14| HOARE, C. A. R. - "Communicating Sequential Processes" - Commun. ACM, 21, 8, (Aug 1978), pg 666-677.

|15| BRINCH HANSEN, P. - "Distributed Processes: A Concurrent Programming Concept" - Commun. ACM, 21, 11, (Nov 1978), pg 934-941.

|16| LE LANN, G. - "Distributed System - Towards a Formal Approach", Information Processing 77, Ed. B. Gilchrist, IFIP, North-Holland Publishing Company, 1977, pg 155-160.

|17| ZIMMERMANN, H. - "ISO Reference Model - The ISO Model of Architecture for Open System Interconection", IEEE Trans. Comm., (April 1980), pg 425-432.

|18| TANENBAUM, A. S. - "Computer Networks", Prentice-Hall, NJ, 1981.

|19| LAUER, H. C. e NEEDHAM, R. M. - "On the Duality of Operating Systems Structures" - Poceedings of the 2nd International Symposium on Operating Systems, IRIA, (Oct 1978), re-impresso em Operating Systems Review 13, 2 (April 1979), pg 3-19.

|20| NELSON, B. J. - "Remote Procedure Call" - Dissertação de Doutorado - Departament of Computer Science, Carnegie-Mellon University, 1981.

|21| ANDREWS, G. R. e SCHNEIDER, F. B. - "Concepts and Notations for Concurrent Programming" - ACM Comp. Surv, 15, 1, (March 1983), pg 3-43.

|22| WIRTH, N. - "Toward a Discipline of Real-Time Programming" - Commun. ACM, 20, 8, (Aug 1977), pg 577-583.

|23| DIJKSTRA, E. W. - "Guarded Commands, Nondeterminacy, and Formal Derivation of Programs" - Commun. ACM, 18, 8, (Aug 1975), pg 453-457.

|24| ALLWORTH, S. T. - "Introduction to Real-Time Software Design" - The Macmillan Press LTD, London,1981.

|25| MAROVAC, N. - "On Interprocess Interaction in Distributed Architectures" - Computer Architecture News 11, 4, (Sept 1983),

pg 17-22.

|26| SPECTOR, A. Z. - "Performing Remote Operations Efficiently on a Local Computer Network" - Commun. ACM, 25, 4, (April 1982), pg 246-260.

|27| SHRIVASTARA, S. K. e PANZIERI, F. - "The Design of a Reliable Remote Procedure Call Mechanism", IEEE, Transactions on Computers,July 1982

|28| KIRNER, C. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Uma Implementação" - 4 Congresso da SBC, 11 Seminário Integrado de Software e Hardware , 1984.

|29| MORON, C. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Supervisor do Processador de Comunicação" - 4 Congresso da SBC, 11 Seminário Integrado de Software e Hardware, 1984.

|30| TEIXEIRA, C. A. C. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Arquitetura do Processador de Comunicação" - 4 Congresso da SBC, 11 Seminário Integrado de Software e Hardware, 1984.

|31| MARQUES, E. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Implementação do Software Básico de Comunicação". IV Simpósio de Software Básico - São José dos Campos, ITA 1984.

|32| BRAYER, K. & LAFLEUR, V. - "A Testbed Approach to the Design of a Computer Comunication Network" - Computer, october, 1982, pg.14-23.