DESENVOLVIMENTO DE SUPORTE BÁSICO PARA SISTEMAS OPERACIONAIS DISTRIBUÍDOS

Cláudio Kirner

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

Aprovada por:

Profª. Sueli B. Teixeira Mendes
(Presidente)

Prof. Cláudio L. Amorim

Prof. Paulo M. Bianchi França

Prof. José Hiroki Saito

Prof. Valentin Obac Roda

RIO DE JANEIRO, RJ - BRASIL
AGOSTO DE 1986

KIRNER, CLÁUDIO

Desenvolvimento de Suporte Básico para Sistemas Operacionais Distribuídos [Rio de Janeiro] 1986.

XVI, 232 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação (1986).

Tese - Universidade Federal do Rio de Janeiro, COPPE.

1. Sistemas Distribuídos I. COPPE/UFRJ II. Título (série).

Esta tese é dedicada:

- à memória de meus avós, Eduardo Kirner e Olga Dering Kirner, e de meu pai, Rodolpho Kirner, que sempre me apoiaram e incentivaram;
- à minha mãe, Antonietta;
- à Tereza e Janice.

AGRADECIMENTOS

À Professora Sueli Mendes, pela orientação, incentivo e apoio durante o desenvolvimento deste trabalho.

À Professora Lídia Micaela Segre, pela sua participação e colaboração nos estudos dos assuntos relacionados com a tese.

Aos professores, técnicos, e bolsistas de iniciação científica do Departamento de Computação e Estatística da UFSCar, que contribuiram para a implementação dos protótipos dos processadores de comunicação.

À CAPES, pelo auxílio concedido.

À Maria José Gualtieri da Costa, pela datilografia, e à Ana Sigoli Fernandes Matheus pela elaboração das figuras.

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

DESENVOLVIMENTO DE SUPORTE BÁSICO PARA SISTEMAS OPERACIONAIS DISTRIBUÍDOS

CLÁUDIO KIRNER

AGOSTO, 1986

Orientadora: Sueli Mendes
Programa: Engenharia de Sistemas e Computação

Este trabalho trata da infraestrutura necessária para o desenvolvimento de sistemas operacionais distribuídos, apresentando o projeto de um suporte básico formado por um subsistema de comunicação e pelo software básico de comunicação entre processos. Neste contexto, busca-se apresentar os elementos teóricos e práticos para a implementação e uso de uma bancada de trabalho que permita desenvolver sistemas operacionais distribuídos. A elaboração desses sistemas operacionais pode ser bastante facilitada pela abstração da estrutura da rede e dos detalhes de baixo nível, envolvidos na comunicação, o que possibilita atacar-se diretamente as questões típicas do sistema operacional.

Os vários capítulos deste trabalho abordam a caracterização de sistemas distribuídos, os seus aspectos de hardware e de software, e uma análise de núcleo e de sistemas operacionais distribuídos. Além disso, são apresentados os projetos de dois tipos de subsistemas de comunicação e o projeto de um suporte básico de uma rede em laço com conexão ponto a ponto.

A principal contribuição deste trabalho para a área de sistemas distribuídos está na especificação de um núcleo de sistema operacional distribuído que suporta comunicação entre processos com tolerância a falhas.

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

BASIC SUPPORT DEVELOPMENT FOR DISTRIBUTED OPERATING SYSTEMS

CLÁUDIO KIRNER

AUGUST, 1986

Adviser: Sueli Mendes
Department: Engenharia de Sistemas e Computação

The main purpose of this work is to present a basic environment for developing distributed operating systems. The runtime support includes a communication subsystem and a software component that supports process communication. Basically we try to offer theoretical elements and practical aspects for the use and implementation of a testbed that can be considered as a fundamental tool for the building of distributed operating systems. The use of this testbed permites to ignore practically all the low level details, i.e. itens related to machine features and communication routines that implements message switching. As a result of this approach we can directly deal with questions tipically related to operating systems.

The first chapters of this work present an overview of distributed systems, its hardware and software aspects, as well as distributed kernels and operating systems. Besides, two different designs of communications subsystems and a basic support for a point to point loop network are presented.

Our main contribution to the research on distributed system is a specification of a distributed operating system kernel that supports process communications with fault tolerance.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO VII

CAPÍTULO VIII

ÍNDICE DE QUADROS

ÍNDICE DE ALGORÍTMOS

CAPÍTULO I

INTRODUÇÃO

Os sistemas distribuídos têm crescido em importância e utilização, sob a influência de vários fatores, dentre os quais a evolução de tecnologias computacionais de hardware e software e a utilização de mecanismos de cooperação entre processos. A evolução tecnológica vem propiciando a multiplicidade de recursos a custos acessíveis, enquanto que os mecanismos de cooperação entre processos têm sido usados para aumentar eficiência, confiabilidade, e disponibilidade de sistemas.

Devido ao elevado tamanho e complexidade do software de sistemas distribuídos, o seu desenvolvimento envolve, essencialmente, um conhecimento profundo dessa área e a utilização de técnicas adequadas de concepção e projeto de sistemas.

De acordo com PAUL e SIEGERT [125], a produção eficiente de software correto e confiável deverá ser precedida pelo estabelecimento completo e detalhado das especificações do sistema, passando-se em seguida à fase de projeto de seus componentes e à consecutiva implementação. Nesse contexto, a utilização de níveis de abstração também constitui-se num fator favorável ao sucesso do desenvolvimento do projeto de sistemas distribuídos.

A existência de um ambiente adequado, para que o desenvolvimento de sistemas possa ser feito sem o conhecimento dos detalhes de baixo nível, facilita bastante a elaboração de sistemas distribuídos, particularmente de sistemas operacionais distribuídos. Visando a obtenção desse ambiente, procurou-se, neste trabalho, abordar o suporte básico para sistemas operacionais distribuídos.

O suporte básico para sistemas operacionais distribuídos consiste do hardware, correspondente ao subsistema de comunicação de uma rede, e do software básico de comunicação que possibilita a interação entre os vários processos do sistema. A partir do suporte básico, o projetista poderá trabalhar com proces

sos e primitivas de comunicação, abstraindo-se dos detalhes da rede.

Assim, optou-se pela elaboração de suportes básicos, denominados bancadas de trabalho, apropriados ao desenvolvimento de sistemas operacionais distribuídos. Para isto, tomou-se como base os trabalhos de BRAYER e LAFLEUR [20], LINDSAY [102], e TOKUDA e MANNING [188], e deu-se especial importância às atividades de discussão e especificação dos projetos.

As bancadas de trabalho foram formuladas de maneira a conter tanto os elementos adequados à montagem de subsistemas de comunicação, quanto o software básico de comunicação. Além disso, os interesses do projeto convergiram muito mais para as questões relativas ao comportamento do sistema, visando seu uso em ensino e pesquisa, ficando em segundo plano as questões de desempenho. Desta maneira, procurou-se explorar prioritariamente os aspectos relacionados com tolerância a falhas, confiabilidade, e recuperação dinâmica do subsistema de comunicação. Com esse enfoque, a obtenção de sistemas distribuídos com bom desempenho pode ser conseguida através da alteração objetiva de hardware e software de projetos desenvolvidos na bancada de trabalho.

A nível de hardware, desenvolveu-se dois tipos de processadores de comunicação para a elaboração de subsistemas de comunicação: processadores de comunicação para conexão ponto a ponto; e processadores de comunicação para conexão a um barramento paralelo centralizado. Os processadores de comunicação, por terem sido implementados com componentes da família de microprocessadores, apresentam programas supervisores que cuidam de atividades como transmissão e recepção de pacotes, tratam certas falhas, e, em alguns casos, determinam rotas. O uso desses dois tipos de subsistemas de comunicação permite analisar-se comparativamente a implementação de vários mecanismos de comunicação e sincronização e o comportamento de cada um.

A nível de software, tomou-se um subsistema de comunicação com conexão ponto a ponto com topologia em laço, e sobre ele desenvolveu-se o programa supervisor do processador de comunicação, contendo todo suporte necessário para a realização de detecção e recuperação de falhas na rede. Este software aborda questões relacionadas com tratamento de falhas, determinação de rotas, gerenciamento de "buffers", controle de fluxo e de conges

tionamento, etc. Particularmente com relação ao gerenciamento de "buffers", adotou-se a solução de manter os "buffers" no processador origem, remetendo um aviso de mensagem, denominado anúncio, para o processador destino. Para isto, desenvolveu-se um protocolo de comunicação específico que considera fatores como reconhecimento e "time-out" para apoiar o tratamento de falhas.

As primitivas de comunicação são formadas por duas parcelas distintas de programa, ambas contidas no núcleo: uma embutida no programa supervisor do processador de comunicação, e outra implementada no processador operador. A especificação da parte da primitiva situada no processador operador está feita sob a forma de diagrama de estados e abrange as primitivas *envia* e *recebe*, bloqueantes e não bloqueantes, e com detecção e com recuperação automática de falhas.

Como o assunto deste trabalho refere-se a análise e desenvolvimento de suporte básico para sistemas operacionais distribuídos, faz-se inicialmente um estudo geral de sistemas distribuídos e de sistemas operacionais distribuídos, seguido da abordagem de questões de comunicação e sincronização. Para finalizar, apresenta-se os projetos de dois tipos de subsistema de comunicação e o projeto do suporte básico de uma rede em laço com conexão ponto a ponto.

Este trabalho está organizado em nove capítulos, conforme detalhamento explicitado em seguida.

O capítulo II - "Caracterização de Sistemas Distribuídos" apresenta as principais características de sistemas distribuídos, enquadrando-os dentro do contexto de sistemas de processamento paralelo, e ressaltando suas diferenças com relação a outros tipos de sistemas como multiprocessadores, redes, etc. A questão de tolerância a falhas é tratada com uma ênfase especial, devido a sua importância nessa área.

No capítulo III - "Hardware de Multiprocessadores e Redes de Computadores" são discutidas as estruturas de multiprocessadores e redes, determinando as características necessárias para a sua aplicação na elaboração de sistemas distribuídos. Embora a base de sistemas distribuídos seja formada por redes de computadores, encontra-se na literatura muitos exemplos de sistemas distribuídos implementados em ambientes multiprocessadores especiais, os quais são analisados.

O capítulo IV - "Software e Estruturação de Sistemas Distribuídos" contém uma análise, envolvendo os aspectos de distribuição do software e os vários modelos de estruturação de sistemas distribuídos relativos à distribuição física e lógica. Aborda-se também as questões relacionadas com as linguagens para programação concorrente, determinando-se as características e os requisitos essenciais para que elas possam ser utilizadas na programação de sistemas.

O capítulo V - "Sistemas Operacionais Distribuídos" trata da localização do núcleo em um sistema operacional distribuído, definindo suas funções e discutindo questões de projeto e de estrutura. Em seguida, faz-se uma análise geral de sistemas operacionais distribuídos, abordando-se aspectos de estruturação e enfatizando-se especificamente os sistemas operacionais distribuídos implementados em ambientes formados por multiprocessadores e por redes.

No capítulo VI - "Comunicação e Sincronização em Sistemas Distribuídos" aborda-se os fatores associados com a comunicação em uma rede e analisa-se a implementação de mecanismos de comunicação e sincronização com relação a: sincronização; endereçamento; utilização de "buffers"; falhas na comunicação; e uso de portos.

O capítulo VII - "Projeto de Subsistemas de Comunicação" apresenta o desenvolvimento de dois tipos de subsistemas de comunicação: um com conexão ponto a ponto; e outro usando barramento paralelo centralizado. Procura-se mostrar as principais características de hardware e de software dos dois tipos de subsistema de comunicação, de forma a permitir uma análise comparativa dos mesmos.

O capítulo VIII - "Projeto do Suporte Básico de uma Rede em Laço com Conexão Ponto a Ponto usando Detecção e Recuperação Automática de Falhas" contém o desenvolvimento de uma bancada de trabalho baseada num subsistema de comunicação com conexão ponto a ponto. Nesse sentido, discute-se os vários aspectos da rede relacionados com: hardware; falhas; roteamento; formato dos pacotes; protocolo baseado em anúncio; núcleo; etc. Analisa-se a estrutura e o funcionamento do programa supervisor do processador de comunicação, e apresenta-se o desenvolvimento das primitivas de comunicação *envia* e *recebe* para atuar em estruturas que

permitam a detecção de falhas ou a recuperação automática de falhas.

As conclusões finais do trabalho e sugestões para futuras pesquisas são apresentadas no capítulo IV - "Conclusões".

Os apêndices apresentam detalhes de hardware e de software dos processadores de comunicação e do suporte básico de uma rede em laço com conexão ponto a ponto.

Finalmente, tendo em vista que este trabalho abrange uma área bastante ampla e ainda pouco explorada a nível nacional, procurou-se oferecer uma contribuição conceitual baseada num conjunto significativo de referências bibliográficas relacionadas com sistemas distribuídos.

CAPÍTULO II

CARACTERIZAÇÃO DE SISTEMAS DISTRIBUÍDOS

Um sistema de computação pode apresentar vários elementos de hardware e de software distribuídos. Muitas vezes, a distribuição de qualquer um desses elementos do sistema, como os dados, por exemplo, pode caracterizá-lo como sistema distribuído.

No sentido de identificar-se os fatores determinantes de sistemas distribuídos, será feita, em seguida, uma análise detalhada das características desses sistemas, procurando enquadrá-los numa área mais ampla, e ressaltando alguns de seus pontos de vital importância.

II.1 - SISTEMAS DE PROCESSAMENTO PARALELO

Devido à limitação intrínseca de sistemas simples de computação e à necessidade de obter-se sistemas que apresentassem melhores índices referentes à: taxa de saída; flexibilidade; disponibilidade; confiabilidade; etc.; e, por consequência, melhor desempenho; os sistemas de processamento paralelo passaram a ser desenvolvidos (ENSLOW JR. [49]).

Surgiram então, nessa linha, dois tipos de sistemas de computação que permitiam executar tarefas paralelas e cooperativas, consistindo de sistemas logicamente paralelos e sistemas fisicamente paralelos. Enquanto os sistemas logicamente paralelos se caracterizam por intercalar o processamento entre várias tarefas em uma única UCP (unidade central de processamento), os sistemas fisicamente paralelos executam as suas tarefas em múltiplos computadores, ou em múltiplas UCPs, compartilhando os demais recursos de máquina, tais como memória principal e unidades de E/S (entrada e saída).

Na classe de sistemas fisicamente paralelos baseados em

múltiplos computadores, apareceram inicialmente os sistemas que executavam entrada e saída separadas do processamento principal. Esses sistemas usavam computadores independentes e submissão, em separado e por partes, de cada etapa de um processamento. Apareceram também os sistemas compostos por computadores interligados frouxamente através de acesso a periféricos comuns (disco ou fita magnética), atuando como caixa postal. Os sistemas compostos por computadores interligados fortemente através de memória compartilhada, ou de conexão de canais de alta velocidade, surgiram nesse contexto.

Na classe de sistemas fisicamente paralelo baseados em múltiplas UCPs, foram desenvolvidos os sistemas multiprocessadores que, além de utilizarem compartilhamento de memória e de unidades de E/S, apresentam um único sistema operacional integrado.

A evolução dos sistemas baseados em múltiplos computadores levou ao aparecimento das redes de computadores e dos sistemas distribuídos. Os multiprocessadores foram sofisticados pela utilização de várias técnicas de obtenção de compartilhamento, incorporando, inclusive, em alguns casos, memória local às UCPs do sistema. Além desses tipos de sistemas paralelos, existem outros, tais como: Processadores de arranjos e vetores; processadores "pipeline"; processadores a fluxo de dados; etc.

Por haver algumas características comuns entre multiprocessadores e sistemas distribuídos, principalmente com relação ao software, alguns sistemas de multiprocessamento serão objeto de análise neste trabalho.

II.2 - SISTEMAS DISTRIBUÍDOS

As primeiras evidências de redes de computadores apareceram no início da década de 60, com o projeto e implementação de sistemas centralizados, contendo um grande número de terminais espalhados. A partir daí, a evolução tecnológica, possibilitando uma redução nos custos de comunicação e de hardware, aliada a um avanço no estudo de redes, propiciou as condições favoráveis ao surgimento das redes de comunicação de computadores.

Ao mesmo tempo, observou-se também um avanço na área de

redes locais que apresentavam uma sensível redução de custo e aumento de confiabilidade e desempenho, decorrentes da restrição da cobertura geográfica da rede.

Nesse estágio, já se tinha um bom domínio das técnicas de compartilhamento de recursos de comunicação, como canais e processadores de comunicação, enquanto que as técnicas de compartilhamento de recursos de computação, como hardware, programas, banco de dados, encontravam-se em fase inicial de desenvolvimento.

O compartilhamento de recursos de computação pode existir de forma explícita ou implícita. Quando o compartilhamento explícito é utilizado, a rede oferece a possibilidade dos usuários acessarem e manipularem diretamente os recursos remotos. Esse tipo de compartilhamento, por exigir que os usuários conheçam os detalhes dos recursos remotos, apresenta algumas dificuldades. Por outro lado, quando se tratar da utilização de compartilhamento implícito, o acesso aos recursos remotos será feito automaticamente pelo sistema de forma transparente ao usuário.

Os sistemas com compartilhamento de recursos de comunicação e compartilhamento implícito de recursos de computação acabaram, em grande parte, constituindo-se nos sistemas distribuídos.

Os termos: "sistemas distribuídos; processamento distribuído; computação distribuída; processamento de dados distribuído" têm sido definidos de forma diversificada por muitas pessoas. É comum denominar-se sistemas distribuídos como aqueles que possuem terminais inteligentes, ou processadores de entrada e saída, ou processadores "front-end", ou processadores de uso geral interligados através de uma rede.

Para definir-se o que significa realmente sistemas distribuídos, ocorreram demoradas discussões sobre este assunto, em vários "workshops" no final da década de 70 (PETERSON [127], ECKHOUSE JR. et alii [47]). Nessas reuniões, ficou claro que há muitos elementos de um sistema que podem ser distribuídos, tais como: processadores (lógica de processamento); dados; programas; e controle (sistema operacional). Estabeleceu-se que a principal característica, que distingue os sistemas de processamento distribuído dos sistemas de arquiteturas clássicas, está na descentralização do controle.

Um sistema distribuído, segundo ECKHOUSE JR. et alii [47], é formado por um conjunto de módulos, compostos pelo menos por processador-memória, interligados frouxamente através de um subsistema de comunicação de topologia arbitrária. Esse hardware deve oferecer facilidades de comunicação entre processadores e entre processos, os quais, cooperando sob um controle descentralizado, possibilitam a execução de programas de aplicação.

Uma consequência prática da interconexão frouxa se reflete na colocação do controle do processamento nos vários módulos do sistema, que serão denominados nós ou estações. Nessa situação, de acordo com PEEBLES e MANNING [126], nenhum processador poderá coordenar os outros, mas todos deverão cooperar em harmonia como acontece em uma comunidade. A comunicação entre processos localizados em nós diferentes será no mínimo uma ordem de magnitude mais lenta, em comparação com a comunicação entre processos situados no mesmo nó.

A definição de sistema distribuído não implica na distribuição geográfica dos computadores que o compõem, pois a conexão fraca pode existir em ambientes confinados ou espalhados. Assim, um sistema distribuído pode ser implementado, tanto dentro de um gabinete quanto numa área geográfica.

De acordo com SPECTOR [170], um sistema distribuído é definido como um conjunto de nós de processamento, possivelmente heterogêneos, contendo programas de controle adequados. O sistema deve possuir, obrigatoriamente, as características *a* e *b*, e, excepcionalmente, algumas das características situadas entre *c* e *g* mencionadas a seguir:

a) *multiplicidade de nós de processamento* para permitir aumento no desempenho, tolerância a falhas, disponibilidade do sistema, e/ou uso de dados geograficamente dispersos;

b) *mecanismos de comunicação* para suportar comunicações entre os componentes do sistema distribuído;

c) *isolação* entre componentes para suportar controle de diferentes entidades administrativas, tolerância a falhas, e disponibilidade;

d) *possibilidade de expansão* para permitir crescimento incremental;

e) *mecanismos de detecção de erros* para obter disponibilidade e tolerância a falhas;

f) *redundância* para obter disponibilidade e tolerância a falhas, pois, uma vez que os erros tenham sido detectados, a redundância é que permitirá a recuperação;

g) *dispersão geográfica* para permitir que os recursos sejam separados geograficamente.

Do ponto de vista de ENSLOW JR. [51], um sistema distribuído deve contar com cinco ingredientes básicos:

a) *multiplicidade de recursos* de uso geral (físicos e lógicos), cuja concessão a tarefas específicas deve ocorrer de forma dinâmica;

b) *distribuição física desses recursos*, interagindo através de uma rede de comunicação;

c) *existência de um sistema operacional de alto nível* que unifica e integra o controle dos componentes distribuídos;

d) *transparência do sistema*, permitindo que os serviços sejam requisitados somente por nome;

e) *autonomia cooperativa*, caracterizando a operação e interação de todos os recursos.

Outra maneira de enfocar a caracterização de sistemas distribuídos baseia-se em três ítens (PARDO [123]):

a) o hardware básico de um sistema distribuído consistirá de uma rede de computadores. Assim, nesse tipo de sistema, existirão vários elementos de processamento, denominados hospedeiros da rede, interligados por um sistema de comunicação conhecido por subrede de comunicação;

b) a arquitetura física do subsistema de comunicação apresenta pouca importância do ponto de vista lógico do sistema. As características dos cabos, da frequência de transmissão, da topologia, etc., impõem maior impacto em questões como desempenho;

c) software é a palavra chave em sistemas distribuídos. A diferença básica entre uma rede de computadores e um sistema distribuído reside no software de cada um. Em uma rede, a cooperação entre sistemas remotos limita-se aos serviços usuais já padronizados, tais como: transferência de arquivos; correio; acesso a terminais remotos; etc. Em um sistema distribuído, a cooperação entre sistemas remotos é bastante elaborada, uma vez que envolve a implementação de compartilhamento implícito de múltiplos recursos remotos.

De acordo com NELSON [121], as partes de um sistema serão *autônomas*, se estiverem *isoladas* do ponto de vista lógico, e *distribuídas*, se estiverem *separadas*. De maneira mais geral, pode-se dizer que a autonomia está relacionada com isolação, e a distribuição com a separação das partes.

Segundo JONES e SCHWARZ [75], a distribuição, considerada como uma espécie de organização, deve ser enfocada sob dois aspectos: do ponto de vista físico; e do ponto de vista lógico. Do ponto de vista físico, a distribuição existirá, se o sistema for composto de pelo menos dois componentes físicos autônomos de capacidade aproximada. A estruturação lógica de um sistema prevê a existência de dois tipos de componentes: os ativos, que constituem os processos; e os passivos, correspondendo aos objetos que contêm dados ou código. Do ponto de vista lógico, um sistema será distribuído, se cada componente for autônomo a ponto da retirada de um deles não provocar degenerações ou falhas não aceitáveis no sistema. Um sistema distribuído confiável deverá ter uma estruturação que o permita ser tolerante a falhas de hardware e de software.

As características de sistemas distribuídos que fazem com que eles sejam indicados em muitos projetos de sistemas são as seguintes (PETERSON [127]):

a) *crescimento incremental*: um sistema distribuído apresenta facilidades de expansão, ao contrário de outros sistemas que não permitem expansão, a não ser por repetição. A repetição apresenta o inconveniente de produzir dois ou mais sistemas distintos, em vez de um sistema maior;

b) *confiabilidade*: sistemas distribuídos podem ser potencialmente mais confiáveis, devido à multiplicidade e a um certo grau de autonomia de suas partes. É largamente reconhecido que a distribuição física não é tão importante quanto a distribuição lógica. Esta última pode ser implementada, tanto em um único processador quanto em vários processadores localizados num mesmo ambiente ou em ambientes distintos. A distribuição física está mais ligada com questões de desempenho, tempo de resposta, organização do sistema, e principalmente com a possibilidade de ter-se controle explícito sobre o processamento e informações locais;

c) *estrutura*: sistemas distribuídos podem refletir a estrutura organizacional, à qual eles servem;

d) *proteção*: sistemas distribuídos podem oferecer mais segurança do que sistemas centralizados. A segurança resulta muito mais da distribuição lógica do sistema do que da sua distribuição física.

A estrutura de sistemas distribuídos pode ser visualizada em camadas (LORIN [107]). A figura (II.1) mostra a estruturação por camadas de um sistema de aplicação com múltiplos terminais, servindo para indicar as características de um sistema passíveis de distribuição.

A camada mais interna, correspondente ao hardware, representa o nó físico de processamento que suporta todas as outras camadas de software. A primeira camada de software, a envolver o hardware, é o núcleo que se comporta como um sistema operacional básico. Sobre o núcleo situa-se um conjunto de serviços, constituindo o sistema operacional propriamente dito, que forne

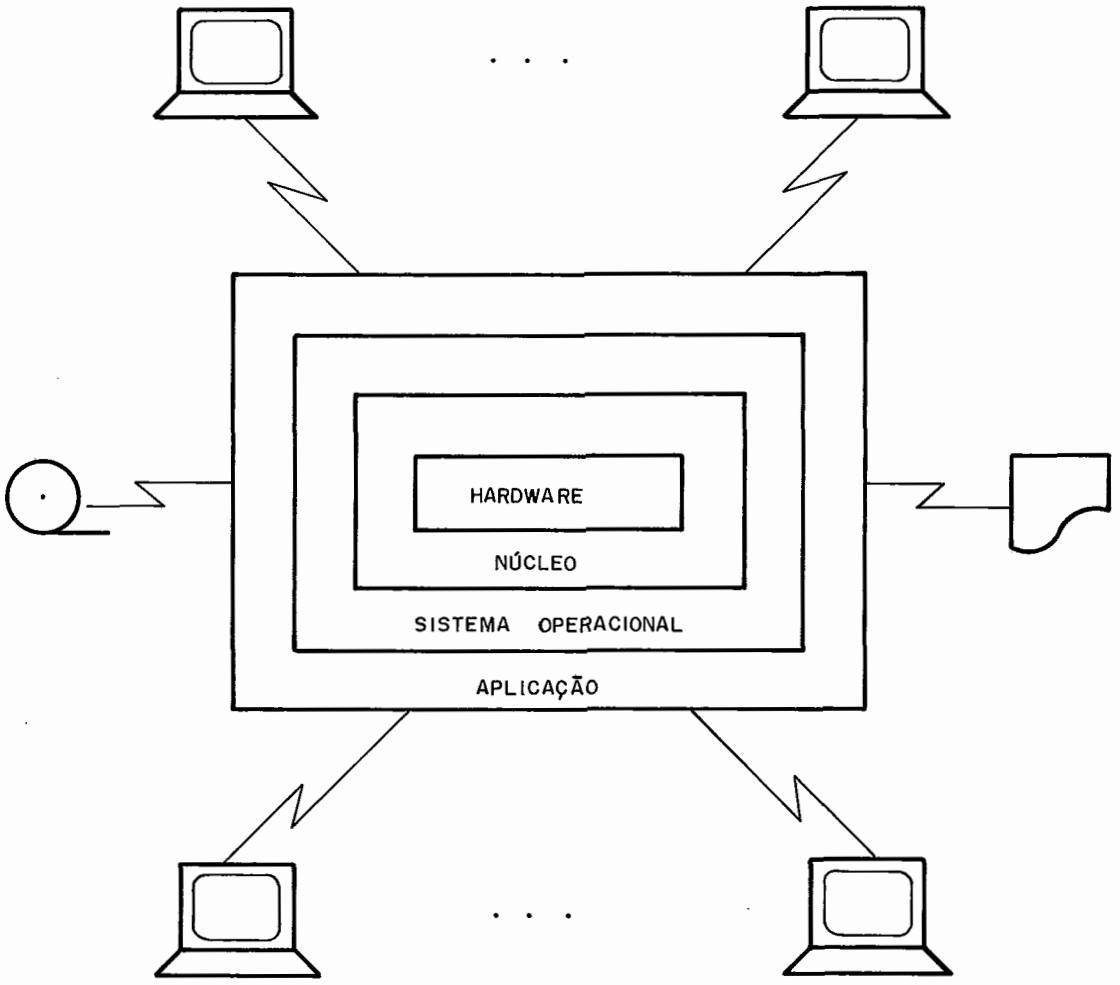


FIGURA II.1 - ESTRUTURAÇÃO DE UM SISTEMA POR CAMADAS

ce o suporte necessário à execução de programas. Em seguida, tem-se a camada mais externa, correspondendo ao nível de aplicação.

Deve-se notar que o núcleo pode estar envolvendo um hardware composto por várias unidades de processamento (unidades centrais de processamento, ou computadores, devidamente interligados), comportando-se, no sentido macroscópico, como uma única máquina.

Dependendo do compartilhamento ou da partição de cada camada, pode-se obter vários tipos de sistemas, alguns bastante próximos aos sistemas distribuídos existentes. Os dois casos ex

tremos: inexistência de partição; e partição de todas as camadas; sugerem uma gama bastante variada de situações intermediárias que encontram aplicações na prática.

A estruturação de sistemas por camadas é muito sugestiva e mostra claramente os três elementos principais que formam a base de sistemas distribuídos e de outros tipos de sistemas: o hardware; o núcleo; e o sistema operacional.

II.3 - ASPECTOS DE TOLERÂNCIA A FALHAS

Pelo fato de sistemas distribuídos possuirem múltiplas partes de hardware e de software funcionando conjuntamente, as chances de alguma dessas partes falhar é bem maior do que ocorreria num sistema simples.

Um sistema distribuído pode apresentar falhas de vários tipos, dentre as quais, aquelas provenientes de defeitos nas linhas de comunicação que interligam os processadores, e aquelas referentes a colapsos nos processadores. Para que um sistema seja confiável, ele deverá continuar funcionando corretamente, mesmos com a presença de certos tipos de defeitos ou interferências indevidas, apresentando portanto tolerância a falhas.

Em sistemas tolerantes a falhas, o efeito de defeitos e interferências indevidas pode ser superado através de redundância temporal e/ou redundância física (SIEWIOREK [159]). A redundância temporal corresponde à determinação de um resultado através de execuções repetidas da mesma operação pelo mesmo elemento, usando eventualmente métodos diferentes. A redundância física consiste em ter-se elementos repetidos capazes de executar a mesma operação. Esses elementos referem-se tanto ao hardware quanto ao software do sistema.

As partes redundantes do sistema podem ser usadas como elementos de votação ou como elementos de reserva. Como elementos de votação, cada elemento redundante permanecerá ativo no sistema, fornecendo seu resultado que, comparado com os outros, contribuirá para a escolha do resultado correto por maioria. Como elemento de reserva, cada elemento redundante permanecerá latente até que seja acionado pelo sistema para entrar no lugar de

outro elemento defeituoso que estiver sendo desativado. Neste último caso, o sistema deverá possuir um bom mecanismo de realização de testes e maneiras eficientes de identificação de defeitos para poder localizá-los e superá-los.

Um sistema, quando não funciona bem, apresenta, como consequência, resultados incorretos conhecidos por *erros*. A causa dos erros é a presença de defeitos ou interferências indevidas, conhecidos por *faltas*, que aparecem tanto a nível de hardware quanto de software. Ao evento, correspondente ao desvio do comportamento esperado do sistema, denomina-se falha (KOHLER [90], BARZAK e AZEVEDO [13], ANDERSON e KNIGHT [5]).

Os sistemas tolerantes a falhas situam-se em duas classes, ou seja, sistemas de alta disponibilidade, e sistemas de alta confiabilidade. A disponibilidade corresponde à probabilidade do sistema de manter-se operacional dentro de um certo período de tempo (SIEWIOREK [159]). A confiabilidade, por sua vez, é o grau de sucesso que um sistema consegue atingir dentro de um período de tempo, mantendo seu comportamento dentro das especificações normais (KOHLER [90]).

Enquanto que sistemas de alta disponibilidade podem tolerar pequenos atrasos decorrentes de manutenção, os sistemas de alta confiabilidade exigem que o funcionamento correto seja mantido, apesar da presença de faltas; pode ser impossível ou inviável qualquer manutenção ou reparo no sistema. Em alguns casos, onde os computadores são instalados em satélites, por exemplo, não há possibilidade prática de realizar-se manutenção; noutros casos, onde os computadores são usados para controle de vôo em aviões, não é possível realizar-se a manutenção durante o vôo, em outras situações, onde qualquer reparo apresenta-se economicamente inviável, a manutenção é contraindicada.

Um sistema poderá apresentar maior confiabilidade, se for desenvolvido com o uso de técnicas adequadas para isso. Há duas abordagens de projeto que conseguem aumentar o grau de confiabilidade de um sistema: a primeira atua no sentido de impedir o aparecimento de faltas; a segunda, no sentido de tolerá-las.

O objetivo principal da abordagem de desenvolvimento de projeto, que procura impedir o aparecimento de faltas, consiste em minimizar o seu aparecimento. Para isto são utilizados componentes de alta confiabilidade, técnicas aprimoradas de projeto

e de verificação, etc. É possível reduzir bastante a ocorrência de falhas no sistema, mas não eliminá-las completamente, motivo pelo qual esses sistemas são conhecidos por intolerantes a falhas.

O objetivo principal da abordagem tolerante a falhas consiste em inibir o efeito das faltas através das redundâncias do sistema. Quando sistemas tolerantes a falhas forem submetidos à ocorrência de uma falha, eles deverão reagir, executando alguns ou todos os procedimentos dos dez estágios seguintes: confinar a falta; detectar a falta; mascarar a falta; tentar de novo; diagnosticar; reconfigurar; recuperar; reiniciar; reparar; e reintegrar (SIEWIOREK [159]).

O confinamento da falta consiste em limitar o alcance de seus efeitos através do uso de: circuitos de detecção de faltas; verificação de consistência, precedendo certas operações; protocolos múltiplos de comunicação; etc.; evitando a propagação das falhas.

A detecção de falta corresponde a tomar conhecimento de sua existência através de bits de paridade, de verificação de consistência, de violação de protocolo, etc. Essa detecção poderá ser feita com o dispositivo fora de operação, ou em operação. Enquanto que, no primeiro caso, a aplicação dos testes será realizada com o dispositivo fora de operação, no segundo caso, os elementos de detecção deverão atuar concorrentemente com o processamento normal, fornecendo capacidade de detecção da falta em tempo real.

O mascaramento da falta tem por objetivo eliminar os seus efeitos, fazendo com que as informações redundantes se imponham sobre as informações incorretas, como acontece, por exemplo, com votação majoritária.

Tentar de novo é um procedimento que dará bons resultados, quando tratar-se de faltas transientes que não provoquem nenhum dano físico ao sistema; uma segunda tentativa poderá ser bem sucedida.

O diagnóstico tem por objetivo identificar a localização da falta e suas propriedades, uma vez que a detecção da falta não trata desse aspecto.

A reconfiguração consiste em substituir ou isolar as partes defeituosas. A opção por isolar as partes defeituosas po

derá levar o sistema a uma degradação elegante (gradual). A substituição das partes defeituosas só será possível, se houver partes correspondentes de reserva.

A recuperação tem a função de colocar a operação do sistema, ou de algumas de suas partes, numa situação anterior à ocorrência das falhas com o objetivo de eliminar os erros que apareceram. Isto exigirá, do sistema, conhecimento de pontos de operação que possam ser restaurados. Algumas técnicas utilizadas para recuperar pontos de operação anteriores consistem em: manter arquivos de "backup"; usar pontos de verificação; manipular listas de intenção; etc.

O reinício, que consiste em recolocar o sistema em funcionamento correto, poderá ser "quente", "morno", ou "frio". O reinício quente, que corresponde à retomada de todas as operações a partir do ponto onde ocorreu a detecção da falta, só será possível, se não tiver ocorrido nenhum estrago nas informações. O reinício morno ocorrerá, quando somente uma parcela do sistema estiver em condições de ser retomada, e a outra parte com informações destruídas deverá ser reiniciada. O reinício frio consistirá do reinício total do sistema, precedido pela recarga, se for necessária, para superar perdas de grande amplitude.

O reparo constitui-se na atividade de consertar o componente defeituoso. Isto poderá ser realizado com o sistema fora de operação ou em operação. No caso de reparo com o sistema fora de operação, o elemento defeituoso deverá ser consertado com o sistema desligado. No caso de reparo com o sistema em operação, o elemento defeituoso deverá ser substituído por um elemento sobressalente nos moldes da reconfiguração, ou deverá ser retirado, ocasionando eventualmente uma degradação elegante no sistema; em qualquer situação, a manipulação do elemento defeituoso não deverá provocar interrupção no funcionamento do sistema.

Os mecanismos de tratamento de falhas num sistema podem diferir bastante entre si, dependendo de quais etapas do tratamento de falhas são utilizadas e da estratégia adotada em cada uma. A figura (II.2) apresenta várias maneiras de realizar o tratamento de falhas em três tipos básicos de sistemas tolerantes a falhas: sistemas com detecção de faltas; sistemas com redundância estática; e sistemas com redundância dinâmica

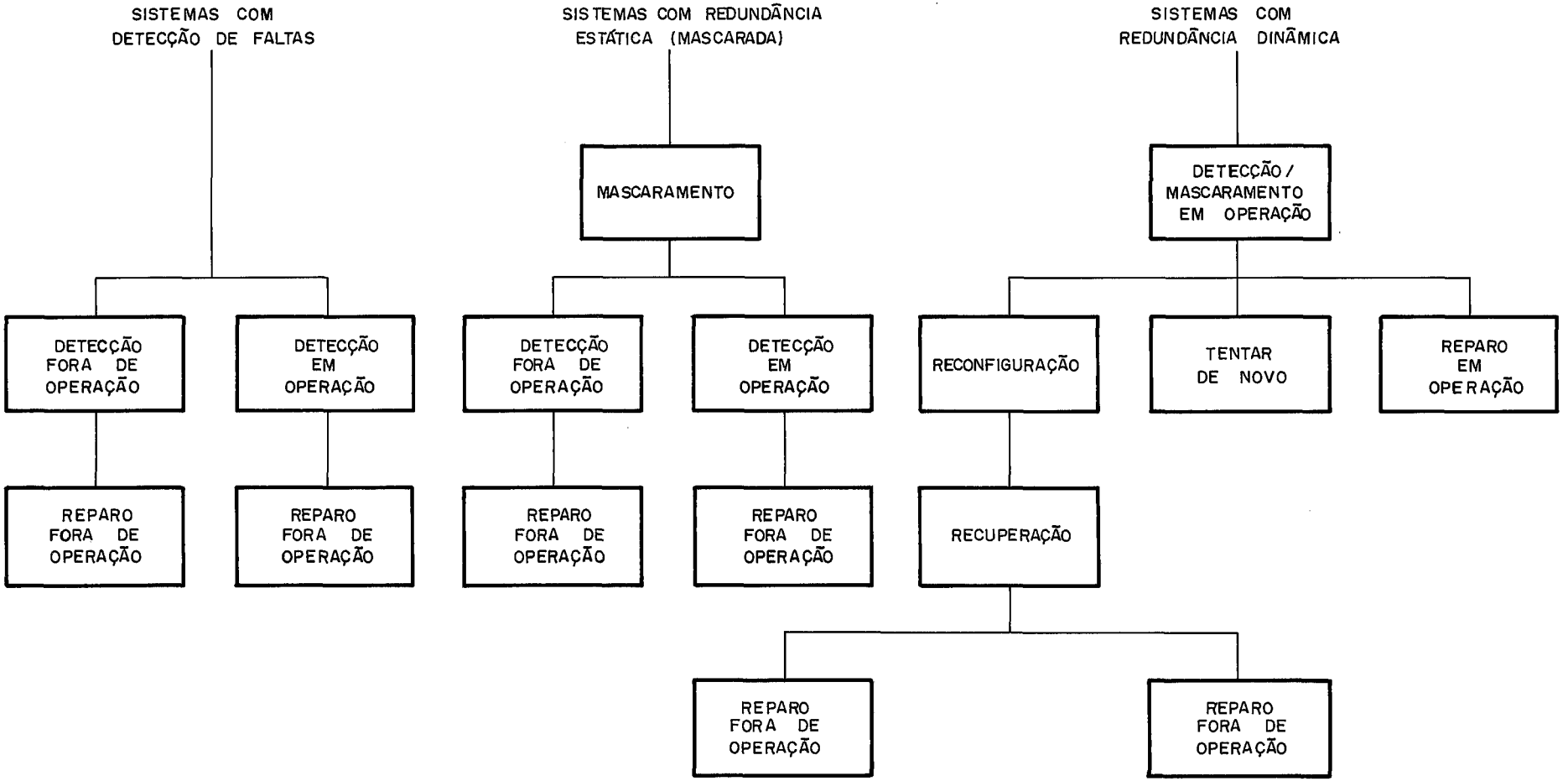


FIGURA II.2 - ESTRATÉGIAS DE TRATAMENTO DE FALHAS EM SISTEMAS TOLERANTES A FALHAS.

(SIEWIOREK [159]).

A rigor, sistemas com detecção de faltas não são tolerantes a falhas, exceto quando estiverem envolvidas faltas transientes que podem ser superadas por novas tentativas de execução. Além disso, a única coisa que podem fazer é avisar o operador que a falta existe.

Os sistemas com redundância estática superam as faltas, mascarando-as, podendo não avisar o operador dessa ocorrência. Esses sistemas utilizam código de correção de erro, redundância com votação majoritária, etc.

Os sistemas com redundância dinâmica superam as faltas, alterando dinamicamente sua própria estrutura. Alguns desses sistemas permitem a realização de reparos em fase de operação.

Em termos de quantidade de redundância de hardware necessária ao bom funcionamento de sistemas tolerantes a falhas, os sistemas com detecção de faltas são os que funcionam com menor redundância, enquanto que os sistemas com redundância estática são os que apresentam maior grau de redundância; os sistemas com redundância dinâmica utilizam um grau médio de redundância.

No desenvolvimento de sistemas tolerantes a falhas, a estruturação cuidadosa do sistema constitui-se num fator tão importante quanto a elaboração de algorítmos eficientes. Nesse sentido, RANDELL [135] e RUSHBY e RANDELL [145] apresentam três maneiras fundamentais de se conseguir uma boa estruturação de sistema: a primeira é baseada no uso de componentes ideais tolerantes a falhas; a segunda refere-se ao uso de estruturação recursiva; e a terceira corresponde ao uso de operações atômicas. Em seguida, essas formas de estruturação são detalhadas.

A estruturação com componentes ideais tolerantes a falhas (de hardware e de software) procura deixar claro quais partes do sistema são responsáveis por quais tipos de faltas, usando para isto um detalhamento adequado dessas partes e de suas relações com o resto do sistema. Um componente ideal tolerante a falhas deve ser capaz de mascarar suas próprias falhas e qualquer outra falha não mascarada que apareça nos componentes que ele faz uso. Assim, o componente ideal se mostra inteiramente confiável ao seu ambiente. Entretanto, isto nem sempre é possível, tornando necessário que componentes tolerantes a falhas de

vam possuir meios pré-definidos de avisar seu ambiente sobre a ocorrência de falhas que, por algum motivo, não possam ser mascaradas.

A estruturação recursiva consiste em considerar-se computadores completos como componentes de um sistema distribuído, de forma que o sistema possa ser funcionalmente equivalente a cada um desses componentes. Isto significa que o sistema pode ser estendido indefinidamente, pelo menos teoricamente, e cada computador deverá ser usado da mesma maneira, estando isolado ou inserido no sistema.

As operações atômicas são aquelas que, sendo executadas por um conjunto de componentes do sistema, não apresentam nenhuma interação com qualquer outro conjunto de componentes. Essas operações apresentam duas propriedades separadas, mas mutuamente dependentes, consistindo da atomicidade e da indivisibilidade (LISKOV [105]). A atomicidade é a propriedade do tudo ou nada, que assegura que, se a operação não puder completar-se inteiramente, ela não terá nenhum efeito. A indivisibilidade, por sua vez, garante que os estados intermediários da operação não serão visíveis às outras operações. Se uma falta, resultando numa propagação de erro e consequente recuperação, acontecer dentro de uma operação atômica, será possível garantir que isto não afetará as outras atividades do sistema. Por outro lado, se as atividades de um sistema puderem ser organizadas como operações atômicas, a tolerância a falhas poderá ser incorporada em cada operação de forma independente. Assim, a criação e destruição dinâmicas de componentes, dentro de operações atômicas, poderão ser realizadas com relativa facilidade. Quando uma operação atômica envolver diversos nós de uma rede, ela deverá ser implementada com o uso de protocolos adequados, visando permitir que, em caso de falha, possa haver abandono da atividade por iniciativa do próprio nó, ou de outros nós, com a consequente recuperação da atividade (KOHEER [90]). Portanto, as operações atômicas constituem-se em estruturas, capazes de encapsular técnicas de tolerância a falhas dentro de componentes modulares.

Além dessas formas de estruturação de sistemas tolerantes a falhas, é importante levar-se em conta as seguintes observações sobre a estrutura geral desses sistemas (BARZAK e AZEVEDO [13]): é recomendável que se utilize uma combinação de técni

cas de prevenção e de tolerância a falhas; deve-se procurar utilizar estruturas simples e modulares, tanto de hardware quanto de software; os circuitos integrados e subsistemas serão mais eficientes, se apresentarem internamente tolerância a falhas; a tolerância a falhas por software pode minimizar a exigência de tolerância a falhas por hardware; na medida do possível, os elementos redundantes de hardware e de software deverão apresentar estruturas lógicas diferentes.

A diminuição de eventuais problemas de desenvolvimento desses sistemas pode ser conseguida com a utilização de abordagens metódicas de projeto, e com a atuação de equipes diferentes na elaboração do projeto e no seu julgamento.

CAPÍTULO III

HARDWARE DE MULTIPROCESSADORES E DE REDES DE COMPUTADORES

Ao se tratar de sistemas distribuídos, percebe-se por um lado, que as redes constituem a sua base, enquanto que por outro lado, os multiprocessadores apresentam certa proximidade com esses sistemas. Por este motivo, serão analisados em seguida os aspectos de hardware e de organização dos multiprocessadores e das redes, procurando evidenciar suas características e diferenças.

III.1 - MULTIPROCESSADORES

Um multiprocessador é definido, conforme ENSLOW JR. [49,50], como um computador que contém dois ou mais processadores com capacidades aproximadas, organizados de forma a compartilharem, tanto uma memória comum quanto canais de E/S e periféricos, tudo isso sob controle de um sistema operacional integrado (figura (III.1)).

Pelo fato dos processadores de um sistema multiprocessador compartilharem os recursos de hardware e apresentarem alta velocidade de comunicação, diz-se que os mesmos são fortemente acoplados.

A classificação de sistemas multiprocessadores baseia-se no subsistema de interconexão, levando em conta sua topologia e forma de operação.

III.1.1 - CONEXÃO DOS PROCESSADORES COM A MEMÓRIA

Os processadores podem ser conectados à memória compartilhada por meio dos seguintes elementos físicos de ligação: bar

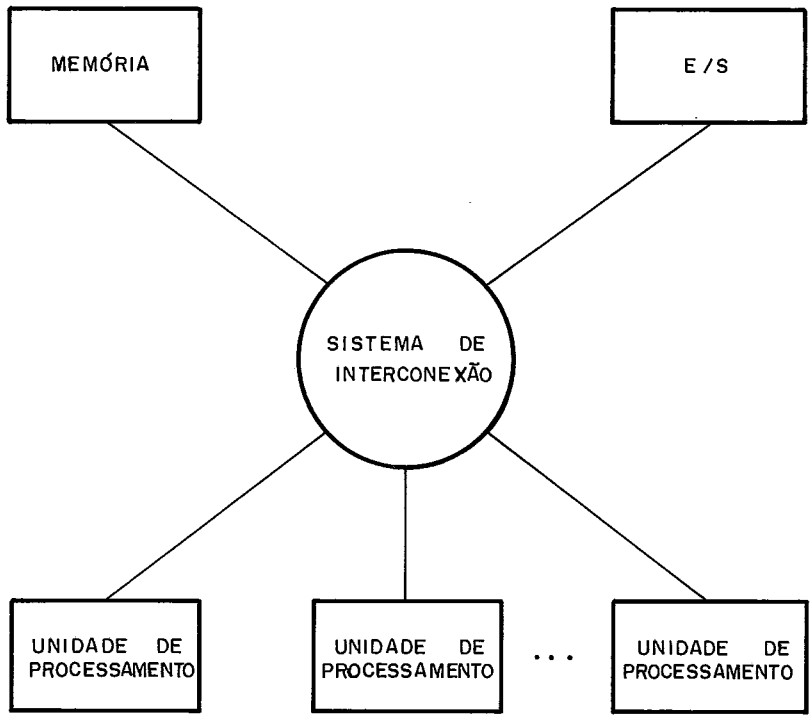


FIGURA III . 1 - ORGANIZAÇÃO BÁSICA DE MULTIPROCESSADORES.

ramento comum; múltiplos portos de memória; comutador centralizado organizado em barras cruzadas; e comutador distribuído (ENSLOW JR. [50], JOSEPH et alii [78]). Além disso, existem outros esquemas, dentre os quais destacam-se aqueles que usam a memória compartilhada sob a forma de caixa postal (WEITZMAN [198]).

A interligação por meio de barramento comum utiliza um meio de comunicação compartilhado pelas unidades de processamento e de memória, consistindo, por exemplo, de um cabo com vários fios. Nesse caso, a unidade de recepção deverá reconhecer seu endereço, atender a requisição formulada, e responder para o emissor. A disputa pelo uso do barramento comum pode ser resolvida de várias maneiras, envolvendo, por exemplo: a utilização de um circuito centralizado, denominado árbitro/controlador do barramento; ou outros métodos rápidos de resolução de disputa.

A utilização de barramentos múltiplos, com os respecti

vos mecanismos de resolução de disputas, servem para aumentar o grau de confiabilidade desses sistemas e torná-los tolerantes a falhas. O número de elementos interligados ao barramento é limitado, em função das características elétricas e das taxas de resposta mínima e média admissíveis para o sistema.

A interligação por meio de múltiplos portos de memória exige que a memória seja dividida em vários bancos, cada um com vários portos. A conexão de cada processador a todos os bancos de memória, através de seus portos, permitirá o acesso concorrente de diferentes processadores a diferentes bancos de memória, mantendo, no entanto, a restrição de acesso serial a um mesmo banco. Cada banco de memória deve apresentar um controlador, que atenderá às múltiplas requisições de acordo com uma política pré-determinada baseada em: ordem de chegada; prioridade dos processadores; ou outra característica específica. Embora simples, esta estrutura de interligação fica limitada pelo número de portos de cada banco de memória.

A interligação por comutador centralizado, organizado em barras cruzadas, baseia-se numa estrutura de conexão completa com relação às unidades de memória. Isto corresponde à existência de um barramento separado para cada unidade de memória, ao qual todos os processadores são ligados sob controle do comutador. Esta estrutura permitirá que as requisições dos processadores sejam, sempre que possível, dirigidas em paralelo para as unidades de memória, ficando a resolução dos conflitos sob responsabilidade do comutador. Além de sua simplicidade conceitual, esta estrutura apresenta a vantagem de poder suportar um número bem maior de processadores, se comparada com os esquemas já citados. Devido à limitação do número de processadores, imposta pela capacidade de atendimento da memória compartilhada, várias maneiras de atenuar os efeitos da disputa pelo acesso a essa memória foram propostas e desenvolvidas. Nesse sentido, além dos múltiplos bancos de memória, que abriram a possibilidade de acesso paralelo a bancos diferentes, outra forma de reduzir as disputas consistiu em usar memória "cache" entre cada processador e o mecanismo de comutação. Isto permitiu que um acesso fosse aproveitado para antecipar informações, desde que fossem tomados alguns cuidados como a atualização da memória, sempre que houvesse alteração em trechos do "cache", ou limitação do mesmo a se

ções só de leitura, envolvendo, por exemplo, instruções e tabelas fixas. Numa abordagem, próxima daquela correspondente à memória "cache", introduziu-se uma memória local para cada processador, de forma a armazenar os dados locais ao processo. Isto provocou, no entanto, uma assimetria na memória do sistema, pelo fato de nenhum processador conseguir acessar toda a memória existente, mas somente a memória compartilhada e sua memória local. A utilização de memória local tem sido muito comum nos projetos mais recentes de multiprocessadores.

Além das limitações de acesso à memória, ocorrem também, no caso de comutadores centralizados, limitações, decorrentes do projeto físico, que impõem um número máximo de interligações, impedindo ou dificultando a expansão do multiprocessador. Para contornar esse problema, visando fazer com que esses sistemas pudessem ter crescimento incremental e maior confiabilidade, desenvolveu-se o esquema de comutador distribuído.

O esquema de comutador distribuído baseia-se na existência de vários comutadores interligados de alguma maneira, cada um associado a um processador e sua correspondente memória local. Ele é organizado de maneira que cada memória local seja uma parcela da memória compartilhada do sistema, sendo acessada diretamente pelo seu processador associado, ou indiretamente por qualquer outro processador, através dos comutadores distribuídos. Esta estratégia de comunicação é denominada comutação de pacote, contrastando com aquela, usada no comutador de barras cruzadas, conhecida por comutação de circuito. Na comutação de pacote, uma requisição de acesso a qualquer posição de memória localizada fora do espaço da memória local associada é conhecida como requisição de acesso remoto, e será convertida em mensagem, ou em pacote. Esta informação será, então, dirigida ao comutador destino, através de comutadores intermediários que constituem a rota, para que seja feito o acesso direto à posição desejada de memória. Após realizar o acesso, o comutador destino devolverá uma informação para o processador emissor da requisição, que deverá estar esperando pela chegada da resposta. Durante o tempo de espera, o comutador ligado ao processador emissor deverá ficar liberado para manipular requisições de outros processadores, funcionando tanto como elemento intermediário quanto como elemento destino. Esta estrutura, conhecida como multiprocessador distri-

buído, é parecida com redes de computadores. No entanto, a diferença deve-se ao fato de cada processador do multiprocessador poder acessar toda a memória do sistema, enquanto que cada processador da rede só pode acessar a sua própria memória local. Apesar de possuir o inconveniente de impor atrasos variáveis no acesso à memória, dependendo do número de comutadores situados entre o processador requisitante e a memória, esta estrutura apresenta uma grande vantagem por não possuir elementos centralizados que possam atuar como gargalos ou pontos críticos do sistema. A estrutura de comutadores distribuídos tem sido utilizada na implementação de sistemas com memória compartilhada virtual. Nesse caso, um mecanismo tradutor de endereço amplia a capacidade de endereçamento de memória de cada processador para fora da memória local a ele disponível.

Um exemplo típico do uso de comutador distribuído pode ser visto no projeto do multiprocessador distribuído Cm* (SWAN et alii [177]). Esse multiprocessador é formado por módulos, contendo microprocessador, barramento local, memória, periféricos, e um comutador local (SLOCAL). Esses módulos são agrupados em conjuntos de no máximo 14 elementos interligados entre si através de um barramento. Cada grupo apresenta um comutador de grupo (KMAP) interligado com os outros comutadores de grupo. Os comutadores locais e os comutadores de grupo atuam coletivamente como um comutador distribuído de memória, conduzindo as requisições de acesso externo à memória aos locais apropriados. Esta estrutura comporta-se como se tivesse uma única memória compartilhada com tempos de acesso diferentes, dependendo da posição acessada.

No sentido de minimizar as referências à memória compartilhada e as consequentes disputas, um multiprocessador poderá ser estruturado de forma que seus processadores tenham memória local exclusiva bastante utilizada, deixando que a memória compartilhada seja usada como um centro de troca de mensagens, funcionando como caixa postal. Nesse caso, os processadores emissores colocariam as suas mensagens em áreas bem definidas ou devidamente identificadas, e os processadores receptores, mediante inspeção, fariam a sua coleta. Este esquema pode ser mais vantajoso do que aquele que usa memória virtual, uma vez que não apresenta sobrecarga adicional com tradução de endereços associada

às referências à memória. Como exemplos desse tipo de implementação tem-se o sistema MIDISS e o comutador PACUIT, citados por WEITZMAN [198], onde a função da memória compartilhada restringe-se exclusivamente a servir como caixa postal para a comunicação entre processadores.

III.1.2 - CONEXÃO DOS PROCESSADORES COM DISPOSITIVOS DE E/S

A conexão dos processadores com os dispositivos de E/S, de forma que estes sejam compartilhados, pode ser feita de várias maneiras, muitas delas muito parecidas com a conexão com a memória compartilhada. Esse tipo de conexão pode dar-se através dos seguintes elementos de ligação: múltiplos portos ligados a um periférico; comutador de barras cruzadas; processador ligado aos periféricos locais, funcionando como servidor de E/S; e comutador distribuído, onde o periférico pode ser acessado por qualquer processador local ou remoto.

No esquema de periférico com múltiplos portos, a ligação do periférico com os processadores é feita pelos portos que conduzem e recebem informações correspondentes a dados e controle. Este esquema é pouco usado, e muito raramente incorporado no projeto de controladores de periféricos.

A ligação de processadores com periféricos, através de comutador por barras cruzadas, é semelhante àquela feita com memória compartilhada, embora o comutador de E/S seja bem mais complexo. Esta complexidade existe porque o periférico precisa ficar ligado ao processador durante a transação,bem mais demorada que a da memória. Diante disso, as disputas que envolverem atrasos consideráveis serão mais convenientemente resolvidas a nível de software, em vez de usarem o comutador centralizado.

O esquema de processador servidor de periféricos é o mais comum. O acesso remoto aos periféricos só poderá ocorrer por meio de software, mediante o encaminhamento de requisições de E/S dos processadores "clientes" para o processador servidor de periféricos.

A utilização de comutador distribuídos está relacionada com o acesso dos periféricos, usando endereços de memória no lugar de instruções de E/S, de forma que qualquer operação de E/S

seja iniciada pela escrita de algum dado em determinadas posições de memória. Assim, o comutador distribuído usado para os acessos à memória não precisa ser alterado ou estendido para permitir o acesso aos periféricos. Embora todos os processadores tenham capacidade de disparar operações de E/S, eles acabam deparando-se com sérias dificuldades para poder acompanhá-las e controlá-las, devido aos atrasos significativos e variáveis impostos pelos elementos intermediários. Para superar esses problemas, acaba-se deixando as transferências de dados e a manipulação dos sinais de controle por conta do processador, onde o periféricos estiver conectado.

III.2 - REDES DE COMPUTADORES

Geralmente, uma rede de computadores é definida como um conjunto de computadores autônomos (hospedeiros) interligados por um sistema de comunicação, envolvendo linhas de comunicação (figura (III.2)). A autonomia dos processadores e a velocidade mais baixa de comunicação entre eles, imposta pela interligação através das linhas de comunicação, fazem com que os processadores que compõem a rede sejam frouxamente acoplados.

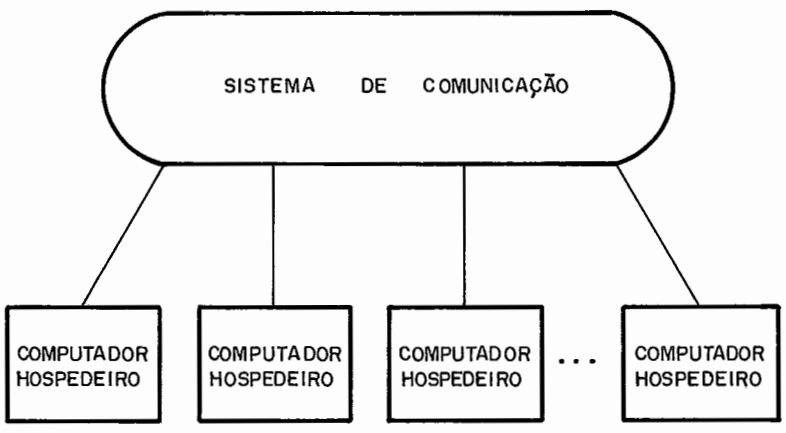


FIGURA III . 2 - ESTRUTURA BÁSICA DE REDE DE COMPUTADORES

O termo "rede de computadores" é comumente utilizado para denominar tanto rede de comunicação de computadores quanto rede de computadores propriamente dita (ELOVITZ e HEITMEYER [48]). Do ponto de vista do hardware, ambos os casos são idênticos, mas, com relação ao software, há uma grande diferença: a rede de comunicação de computadores caracteriza-se por preservar os sistemas operacionais dos computadores hospedeiros, e por exigir do usuário o conhecimento tanto da localização dos recursos na rede quanto da maneira de manipulá-los; a rede de computadores, por sua vez, caracteriza-se por apresentar um único sistema operacional integrado encarregado do gerenciamento do conjunto de recursos, aliviando o usuário da necessidade de conhecer os detalhes dos diversos elementos do sistema. Os sistemas distribuídos, constituindo uma evolução e refinamento do conceito de rede de computadores, acabaram por abranger este tipo de rede, enquanto que rede de comunicação de computadores passou a ser chamada simplesmente por rede de computadores.

Numa rede, o sistema de comunicação que proporciona a interligação dos computadores hospedeiros recebe várias denominações, dentre elas: subsistema de comunicação; ou subrede (DAVIES et alii [42], THURBER e MASSON [185]). O subsistema de comunicação é o elemento determinante na estrutura de uma rede, enquanto os computadores hospedeiros restringem-se, na sua maior parte, aos aspectos de aplicação.

Um subsistema de comunicação é composto pelos elementos de comutação e pelos meios de transmissão. Os elementos de comutação, constituindo-se de interfaces ou processadores específicos, recebem várias denominações tais como: computador de comunicação; comutador de pacotes; nó de comunicação; etc. Os meios de transmissão são conhecidos por circuitos ou canais de transmissão.

Os subsistemas de comunicação utilizam, na sua maioria, um dos seguintes tipos de canais: canais de conexão ponto a ponto; e canais de difusão (TANENBAUM [178]).

Os canais de conexão ponto a ponto correspondem aos circuitos e cabos que permitem a comunicação entre pares de nós da rede. Quando não existir interligação direta entre dois nós, a comunicação entre eles deverá ocorrer através de nós intermediários de forma indireta. Por esse motivo, a topologia da subrede

é de fundamental importância, podendo apresentar-se em estrela, laço, árvore, conexão completa, conexão irregular, etc. A subrede que utiliza esse tipo de canal costuma ser chamada de subrede de ponto a ponto, ou armazena e envia.

Os canais de difusão constituem uma estrutura, onde um único canal de comunicação é compartilhado por todos os nós de comunicação da subrede (ligação multiponto). Esse tipo de sistema deve contar com um mecanismo de arbitragem para resolver conflitos de acesso ao meio comum, determinando a estação que poderá transmitir informações num dado momento. O mecanismo de arbitragem pode ser centralizado ou descentralizado, e a interligação dos nós pode ser feita por cabo, por sistema de ondas de rádio, ou por algum meio de propagação rápida. As subredes que utilizam canais de difusão podem ser classificadas como estáticas ou dinâmicas, conforme a forma de concessão do uso do canal. Na subrede estática, cada nó terá direito a uma parcela de tempo, quando chegar a sua vez, mesmo que não queira transmitir, enquanto que, na subrede dinâmica, o funcionamento é baseado em demanda; neste caso, os nós emitirão requisições para usar o canal, e um árbitro centralizado ou descentralizado dicidirá quem irá usar.

III.2.1 - SUBSISTEMAS DE COMUNICAÇÃO

O custo e desempenho de uma rede estão associados a determinadas características do subsistema de comunicação, dentre elas: topologia; meio de transmissão; e protocolo de controle de acesso ao meio.

III.2.1.1 - TOPOLOGIA

A topologia de um subsistema de comunicação pode ter muitas variações, dependendo das decisões de projeto a respeito de: estratégia de transferência; método de controle da transferência; e estrutura do meio de transmissão. A estratégia de transferência adotada poderá usar transferência direta ou indireta de mensagens. O método de controle da transferência só existirá no caso de transferência indireta, e deverá utilizar roteamento

centralizado ou descentralizado. Finalmente, a estrutura do meio de transmissão poderá apresentar-se sob a forma de meios dedicados ou meios compartilhados.

Tendo essas características como referência, foi montada a taxonomia de arquitetura de subsistemas de comunicação, conforme a figura (III.3), baseada em WEITZMAN [198] e ANDERSON e JENSEN [4].

Cabe citar que, de forma geral, os subsistemas que utilizam meios dedicados tendem a servir-se de canais de conexão ponto a ponto, enquanto que aqueles que utilizam meios compartilhados tendem a servir-se de canais de difusão. Há algumas exceções como, por exemplo, o uso da memória compartilhada como meio compartilhado.

Um quadro comparativo das tecnologias de interconexão, referindo-se a atributos de projeto como confiabilidade, custo, distribuição geográfica, desempenho, etc, é apresentado por WEITZMAN [198]. Este quadro revela uma tendência no sentido de agrupar os subsistemas de comunicação em duas classes com diferenças marcantes relacionadas com esses atributos. Os subsistemas que utilizam canais ponto a ponto, enquadrados numa classe, tendem a ser mais confiáveis, mais caros, ilimitados geograficamente, e de baixo desempenho. Os subsistemas que utilizam canais de difusão, enquadrados na outra classe, possuem uma tendência contrária a anterior. Cabe salientar que os canais ponto a ponto apresentam melhor confiabilidade, pelo fato das falhas, quando ocorrerem, comprometerem somente partes do circuito, enquanto que uma falha num canal de difusão poderá comprometer todo o subsistema. A introdução de redundância nos subsistemas baseados em canais de difusão, dotando-os com tolerância a falhas, altera drasticamente a análise anterior.

III.2.1.2 - MEIOS DE TRANSMISSÃO

Os meios de transmissão utilizados em subsistemas de comunicação são bastante diversificados, variando principalmente em função das dimensões do sistema distribuído. Para sistemas confinados em placas de circuito impresso, ou dentro de gabinetes, onde as distâncias são muito pequenas, os meios de trans

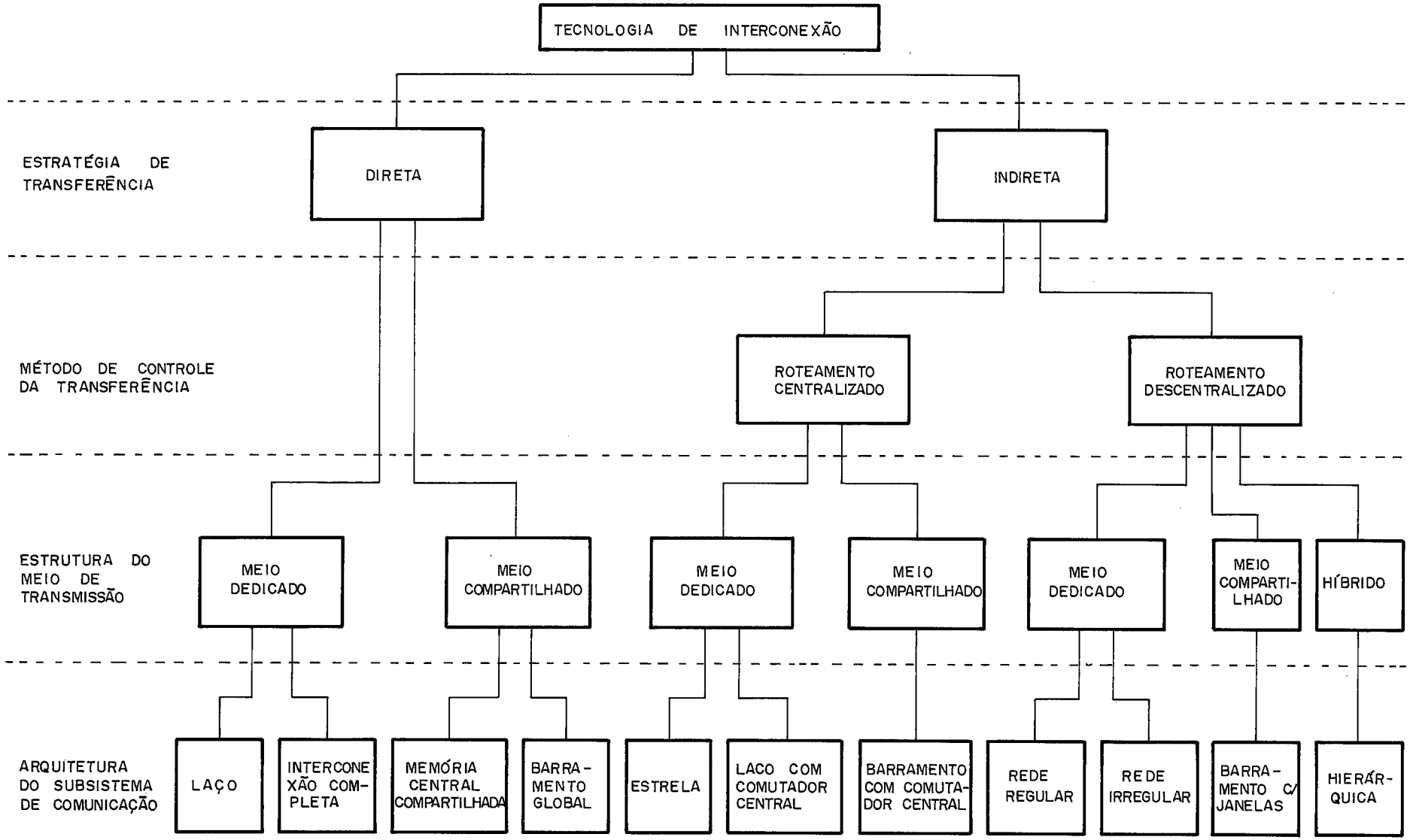


FIGURA III.3 - TAXONOMIA DA ARQUITETURA DE SUBSISTEMAS DE COMUNICAÇÃO

missão são as fiações impressas, ou fios paralelos, interligando essas placas. Os multiprocessadores utilizam-se desses meios de transmissão. Para sistemas, envolvendo distâncias pequenas e médias, os meios de transmissão mais comuns são: par de fios trançado; cabo coaxial; e fibra ótica. Estes meios são utilizados tipicamente em redes locais. Para sistemas geograficamente distribuídos, envolvendo grandes distâncias, os meios de transmissão mais comuns são as linhas telefônicas e sistemas de microondas, utilizando até satélites. As grandes redes geográficas e redes públicas são usuárias desses meios de transmissão.

Tendo em vista que os subsistemas de comunicação, envolvendo distâncias pequenas e médias, tendem a apresentar velocidades e características adequadas a sistemas distribuídos, os meios de transmissão que se ajustam a essa situação serão analisados em seguida.

Cabe citar que esses subsistemas de comunicação utilizam, na sua maioria, canais de difusão, embora existam alguns casos de uso de canais de conexão ponto a ponto, como acontece na configuração estrela. Nesse sentido, os meios de transmissão podem ser analisados, conforme a sua aplicação em canais de difusão de alta velocidade, e em canais de conexão ponto a ponto (STALLINGS [171,172]).

O par de fios trançado, usado tipicamente em transmissões de baixas velocidades, pode atingir taxas de transmissão bidirecional de sinais digitais de até poucos megabits por segundo, em distâncias de poucos quilômetros, e interligando algumas dezenas de dispositivos. Sua principal desvantagem está na suscetibilidade a ruídos e interferências que pode ser minimizada com blindagens adequadas.

O cabo coaxial com características bem superiores as do par de fios trançado pode empregar transmissão em banda base e em banda larga. A principal diferença desses dois tipos de transmissão consiste no fato da banda base trabalhar com um único canal de dados, enquanto que a banda larga abrange múltiplos canais de transmissão simultânea de dados, obtidos através de multiplexação por divisão de frequência.

Os cabos coaxiais que trabalham com banda base podem atingir taxas de transmissão bidirecional de sinais digitais de até 10 megabits por segundo, em distâncias inferiores a 1 quilô

metro, e interligando até uma centena de dispositivos. A principal desvantagem desse meio está na atenuação dos sinais, especialmente em altas frequências de transmissão. A restrição drástica da distância e do número de dispositivos interligados pode elevar a taxa de transmissão a até 50 megabits por segundo.

Os cabos coaxiais, funcionando com banda larga, transmitem sinais analógicos unidirecionais com taxas de transferência de até 20 megabits por segundo, em distâncias de dezenas de quilômetros, e interligando centenas ou milhares de dispositivos. A desvantagem principal desse meio está na sua complexidade de instalação e manutenção. Existem situações, onde o canal unitário de banda larga é usado com sinais analógicos bidirecionais, alcançando taxas de até 50 megabits por segundo, em distâncias menores que 1 quilômetro, e envolvendo algumas dezenas de dispositivos.

A fibra ótica possui uma capacidade potencial superior a do cabo coaxial, funcionando com sinais analógicos transmitidos a uma taxa máxima de 10 megabits por segundo, em distâncias menores que 1 quilômetro, e interligando algumas dezenas de dispositivos. Sua vantagem principal é a alta imunidade a ruídos, embora ainda apresente problemas, devido a custo e limitações técnicas. Uma dessas limitações refere-se à dificuldade de implementação de canais de difusão multipontos com fibra ótica.

O quadro (III.1) mostra uma síntese das características dos vários tipos de canais de transmissão que se utilizam dos meios de transmissão analisados (STALLINGS [171]).

III.2.1.3 - PROTOCOLOS DE CONTROLE DE ACESSO AO MEIO

Nos casos em que vários dispositivos devem compartilhar a capacidade de transmissão dos canais, como acontece com os canais de difusão, a existência de alguma forma de controle de acesso ao meio de transmissão é fundamental para que haja comunicação entre pares de dispositivos interligados por esse meio (STALLINGS [171], KUROSE et alii [94]).

É importante determinar-se *onde* o controle de acesso deve ser localizado, e *como* ele deve funcionar. A questão da localização está relacionada com as discussões de sistemas centrali

Características	Canais de Difusão Normais	Canais de Difusão de Alta Velocidade	Canais de Conexão Ponto a Ponto
Meio de Transmissão (típico)	par trançado de fios cabo coaxial fibra ótica	cabo coaxial	par trançado de fios
Topologia mais usada	barramento anel	barramento	estrela
Velocidade de Transmissão (Mbps)	1 a 20	50	0,0096 a 0,064
Distância Máxima (Km)	25	1	1
Técnica de Comutação (típica)	pacotes	pacotes	circuito
Número de Dispositivos Interligados	centenas a milhares	dezenas	centenas a milhares
Custo Relativo da Ligação	2 a 20	160 a 200	1 a 4

QUADRO III.1 - CARACTERÍSTICAS TÍPICAS DOS CANAIS DE TRANSMISSÃO

zados e distribuídos. Nesse sentido, o controle de acesso deverá estar num local bem definido, ou encontrar-se espalhado nos elementos conectados ao meio de transmissão, apresentando as vantagens inerentes a cada caso. A questão da forma de funcionamento do controle de acesso ao meio está ligada com topologia, custo, desempenho, e complexidade.

O controle de acesso ao meio envolve sempre algum tipo de multiplexação, conforme a classificação da figura (III.4), que pode ocorrer por divisão de frequência ou por divisão de tempo. A multiplexação por divisão de frequência faz com que o meio de transmissão comporte, na prática, múltiplos canais simultâneos. A multiplexação por divisão de tempo de um único canal implica no uso intercalado do meio de transmissão, através de várias técnicas síncronas ou assíncronas.

A multiplexação síncrona por divisão de tempo consiste num método de acesso livre de colisões, que determina o direito de transmissão das estações de maneira estática e pré-determinada. Periodicamente, cada estação ganha um intervalo de tempo bem definido, onde poderá, ou não, realizar transmissões. Por ser ineficiente, este método só poderá ser usado em canais de difusão com poucas estações, ou em canais de conexão ponto a ponto, onde se queira o compartilhamento na ligação.

A multiplexação assíncrona por divisão de tempo é largamente utilizada em canais de difusão, devido a necessidade de atendimento dinâmico de várias requisições de acesso ao meio de transmissão. Este atendimento poderá ser feito de forma aleatória, ou controlada por meio de um algorítmo que determine tanto a sequência dos acessos como o tempo em que possam ocorrer.

A multiplexação por tempo, assíncrona e aleatória, quando aplicada a barramentos, deverá usar o método da disputa, e será obtida através de esquemas de partição probabilística, de partição de endereço, e de partição de tempo. Quando aplicada a anéis, esse tipo de multiplexação será obtido por esquemas de inserção de registros e preenchimento de escaninhos ("slots").

Serão analisados, em seguida, os protocolos de acesso ao barramento e ao anel baseados em diversos esquemas.

— Esquema da Partição Probabilística (Barramento)

O protocolo de controle de acesso ao barramento baseado

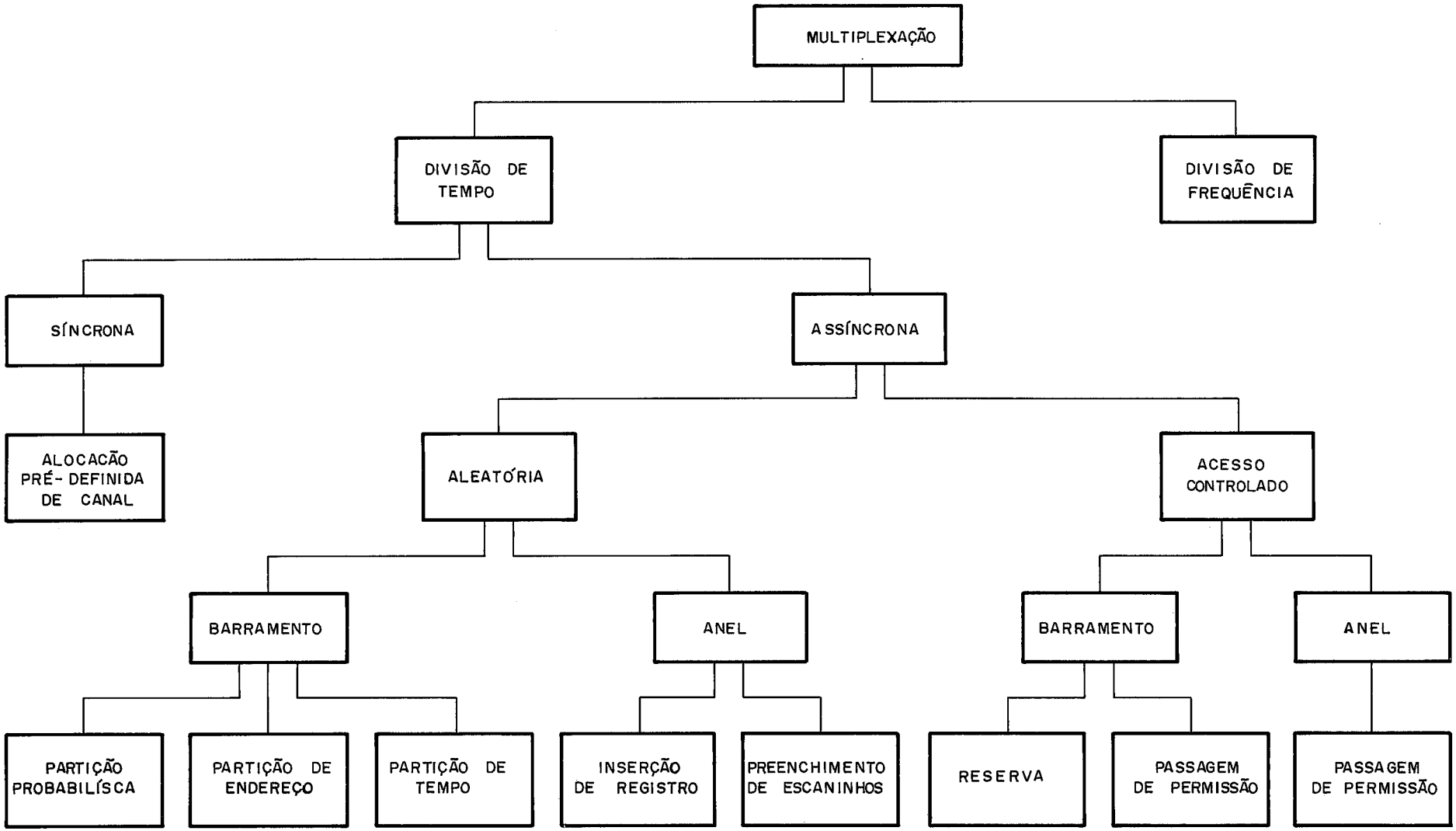


FIGURA III . 4 - TAXONOMIA DOS PROTOCOLOS DE CONTROLE DE ACESSO AO MEIO DE TRANSMISSÃO

no esquema da partição probabilística prevê, em princípio, que uma estação possa transmitir tão logo esteja pronta. Se acontecer que duas ou mais estações transmitam ao mesmo tempo, haverá colisão das mensagens, e elas deverão ser retransmitidas mais tarde, depois de tempos aleatórios de atraso para cada estação. A utilização de intervalos de tempo, onde as estações possam transmitir mensagens, melhora esse protocolo, garantindo que as possíveis colisões ocorram sempre a partir do início da transmissão, e não em qualquer ponto, como podia ocorrer anteriormente. Uma evolução nesse protocolo consiste em dotar a estação com capacidade para sentir (ouvir) o meio de transmissão, de forma a iniciar uma transmissão somente quando o meio não estiver sendo usado. Aqui, tem-se o protocolo de acessos múltiplos com detecção de portadora (CSMA), que evita que uma estação interfira na transmissão de outra, mas não resolve o problema de colisão de duas ou mais estações que começarem a transmitir ao mesmo tempo. Para minimizar o efeito das colisões, existe o protocolo de acessos múltiplos com detecção de portadora e detecção de colisão (CSMA/CD), que funciona como o CSMA, mas interrompe a transmissão em andamento, quando detecta a colisão, e faz com que novas tentativas de transmissão ocorram depois de períodos de tempo aleatórios. Quando uma estação estiver pronta para transmitir, mas o meio estiver ocupado, uma, dentre três possíveis abordagens envolvendo algorítmos de persistência, poderá ser usada. Estas três abordagens são as seguintes:

a) *não persistência:* a estação esperará durante um tempo aleatório, e depois voltará novamente a sentir o meio de transmissão:

b) *persistência-1:* a estação continuará a sentir o meio de transmissão até ele ficar ocioso, quando então fará a sua transmissão;

c) *persistência-p:* a estação continuará a sentir o meio até ele ficar ocioso, quando então, de acordo com uma probabilidade p atribuída previamente, ela poderá transmitir. Se não conseguir transmitir, a estação esperará durante um período de tempo, a partir do qual passará a sentir o meio, repetindo o processo.

— Esquema da Partição de Endereços (Barramento)

O protocolo de controle de acesso ao barramento baseado no esquema de partição de endereços utiliza os endereços das estações para determinar qual, daquelas que estiverem competindo pelo acesso ao barramento, terá direito de realizar a transmissão. O tempo será dividido em períodos bem definidos, e, no início de cada período, todas as estações poderão tentar transmitir; se houver colisão, as transmissões serão suspensas e as estações serão divididas em dois grupos sob a forma de árvore binária, de maneira que somente as estações de um grupo selecionado poderão tentar a transmissão. Enquanto as colisões continuarem acontecendo, haverá sucessivamente a subdivisão do grupo selecionado em dois outros grupos e a escolha de um deles, até que se obtenha um grupo, onde a transmissão ocorra sem colisão. Se houver a seleção de um grupo, onde nenhuma estação esteja querendo transmitir, este grupo será deixado de lado e a tentativa ocorrerá no grupo complementar. Para evitar determinismo na seleção, o que implicaria na existência de estações prioritárias, utiliza-se seleção aleatória das partes da árvore binária. A divisão das estações em grupos e sua escolha podem sofrer muitas variações.

— Esquema da Partição de Tempo (Barramento)

O protocolo de controle de acesso ao barramento baseado no esquema de partição de tempo utiliza a marcação dos momentos, em que ocorrem as requisições de transmissão, para determinar qual estação deverá transmitir. No início de períodos de tempo bem definidos, quando o barramento estiver ocioso, todas as estações selecionarão uma janela de tempo w, a partir de um tempo t_0, e aquelas que, durante o período entre t_0 e t_0+w, fizerem requisições de transmissão, poderão transmitir a partir do momento t_0+w. Se não ocorrer nenhuma requisição nessa janela de tempo, t_0 será atualizado e as estações selecionarão uma nova janela; se houver uma única estação, requisitando transmissão nessa janela, a transmissão ocorrerá normalmente, e, no final, t_0 será atualizado e haverá a seleção de nova janela; se houver duas ou mais estações que tiverem emitido requisições na mesma janela, então elas transmitirão simultaneamente e haverá colisão. No caso

de ocorrer colisão, a janela será dividida em duas metades, ficando a metade mais antiga destacada para funcionar como janela, e assim sucessivamente até que se consiga transmissão sem colisão; depois, ocorrerá a atualização de t_0 baseada no último intervalo de tempo que funcionou como janela. Se uma metade de intervalo de tempo que funciona como janela não contiver nenhuma requisição de transmissão, a metade complementar será selecionada para continuar as tentativas de transmissão. Deverá existir alguma forma de realizar pequenas alterações aleatórias no tempo de requisição para contornar situações de impasse, quando pequenos intervalos de tempo forem atingidos e continuar havendo colisão.

– Esquema de Inserção de Registros (Anel)

O protocolo de controle de acesso a anéis baseado na inserção de registros exige que, associado a cada estação, exista um registrador de deslocamento do mesmo tamanho do maior pacote que circular pelo anel. De maneira complementar, cada estação deverá ter um "buffer" para armazenar os pacotes de informação produzidos. Considerando que o registrador de deslocamento tenha um ponteiro de saída fixo em uma de suas extremidades, e um ponteiro de entrada móvel, o registrador estará vazio e o anel estará ocioso, quando os ponteiros de entrada e de saída coincidirem; caso contrário, o registrador estará sendo usado e o anel estará funcionando. Cada pacote terá um campo inicial de endereço destino que servirá para realizar o endereçamento das estações. Quando o anel estiver ocioso e o "buffer" da estação estiver devidamente preenchido com o pacote, o registrador de deslocamento receberá o conteúdo do "buffer", fazendo com que o ponteiro de entrada seja transferido para a outra extremidade do registrador; em seguida, os pulsos de deslocamento serão emitidos. Se não houver alimentação pelo anel, o ponteiro de entrada irá avançando para a outra extremidade do registrador, juntamente com o pacote. Se houver alimentação de pacotes vindos de outras estações, no início ou durante a transmissão, o ponteiro de entrada ficará bloqueado, permitindo a entrada de bits do novo pacote; isto ocorrerá até que o pacote que estiver sendo transmitido saia totalmente do registrador, quando então, se houver necessidade, os bits adicionais do novo pacote acabarão de entrar no

registrador, e o ponteiro de entrada acabará sendo levado ao último bit do pacote. Sempre que o campo de endereço do pacote atingir o ponteiro de saída, ou vindo por deslocamentos atrás de um pacote que estiver saindo, ou sendo colocado bit a bit a partir do ponteiro de saída (no caso de registrador vazio), esse campo será analisado pela estação. Se o endereço for o da estação, ela verificará se o pacote já está totalmente carregado no registrador de deslocamento, e, em caso afirmativo, fará a retirada de seu conteúdo e a transferência do ponteiro de entrada para junto do ponteiro de saída; caso contrário, esperará o seu preenchimento total do registrador para tomar essas providências. Se o endereço não for o da estação presente, o pacote será deslocado pelo anel e o ponteiro de entrada será manipulado convenientemente. Deverão existir formas de contornar falhas de endereçamento, e de evitar que uma estação monopolize a utilização do anel.

- Esquema do Preenchimento de Escaninhos (Anel)

O protocolo de controle de acesso a anéis baseado no preenchimento de escaninhos ("slots") prevê a existência de um número de escaninhos de tamanho fixo, circulando pelo anel, capazes de transportar pacotes de informação. Os escaninhos possuem bits de controle para indicar tanto se estão cheios ou vazios quanto o tipo de resposta da estação destino, informando se aceitou o pacote, se está ocupada, ou se rejeitou o pacote. O anel será iniciado com todos os pacotes vazios, e uma estação que quiser transmitir deverá esperar pela passagem de um escaninho vazio, quando então fará o seu preenchimento e colocará seu bit de controle como ocupado. Enquanto o escaninho preenchido não retornar, a estação não poderá transmitir outros pacotes. Ao passar pela estação destino, o conteúdo do escaninho poderá ser aproveitado, ou não, e o bit de resposta será assinalado. Ao retornar à estação origem, o escaninho, dependendo do bit de resposta, poderá ser esvaziado ou reencaminhado pelo anel. As estações farão a análise do escaninho através dos bits de controle e pelos campos de endereço origem e destino. É importante que existam formas de identificação dos limites de escaninho.

A multiplexação por tempo, assíncrona, e de acesso con

trolado com base em demanda adaptativa, quando aplicada a barramentos, deverá usar esquemas de reserva e de passagem de permissão ("token"); quando aplicada a anéis, deverá usar o esquema de passagem de permissão. Em qualquer situação, a transmissão será precedida por uma etapa de definição dos direitos de acesso ao meio de transmissão.

— Esquema da Reserva (Barramento)

O protocolo de controle de acesso ao barramento baseado no esquema de reserva, funciona através de períodos de reserva, alternados com períodos de transmissão. Durante um período de reserva, cada estação, de acordo com uma ordem pré-definida, terá uma fração de tempo para requisitar o acesso ao barramento. Todas as estações, sentindo o barramento, acompanharão as frações de tempo de cada estação, anotando aquelas que fizerem reserva. Antes do período de transmissão, cada estação que tiver requisitado reserva analisará suas anotações, e, de acordo com um algorítmo igual para todas as estações, poderá concluir se é ou não a sua vez de transmitir. Esse algorítmo poderá levar em conta as prioridades das estações, ou uma sequência previamente definida para atendimento. As requisições, não atendidas numa fase de transmissão, serão acumuladas na etapa seguinte. Uma variação nessa forma de atendimento consiste em inibir as etapas de reserva seguintes, até que todas as transmissões solicitadas sejam realizadas.

— Esquema da Passagem de Permissão (Barramento)

O protocolo de controle de acesso ao barramento baseado no esquema de passagem de permissão enfoca o conjunto de estações como elementos de um anel lógico, onde cada estação conhece a identidade de sua antecessora e de sua sucessora. Quando for utilizada a passagem explícita de permissão, um pacote de controle, correspondendo à permissão, será passado para uma estação específica que poderá aproveitá-lo, ou não. No caso de aproveitamento, a estação fará, em seguida, a transmissão da informação e, no final, o repasse da permissão para a próxima estação. No caso de não aproveitamento, por não querer transmitir, a estação fará simplesmente o repasse da permissão para a próxima estação,

sem gastar o tempo de transmissão. Quando for utilizada a passagem implícita de permissão, a utilização do barramento ou um pequeno período de tempo em silêncio equivalerão às passagens de permissão. Isto poderá ser implementado num sistema, onde cada estação conhecendo a sua ordem no anel virtual acompanhará ou a utilização do barramento, em termos da transmissão de informação, ou a desistência através do silêncio em pequenos intervalos de tempo. Cada transferência, ou pequeno período de silêncio, corresponderá à passagem da permissão implícita para a próxima estação. Esta permissão poderá ser utilizada pela realização de transmissão, ou repassada imediatamente, através do silêncio por um curto período de tempo, e assim sucessivamente. Esses protocolos podem assumir diversas variações, e devem prever formas de tratamento de erros e de recuperação das permissões.

– Esquema da Passagem de Permissão (Anel)

O protocolo de controle de acesso a anéis baseado no esquema de passagem de permissão utiliza um pequeno pacote de controle que circula pelo anel. Inicialmente, a permissão estará liberada, e uma estação, que quiser transmitir, ficará esperando pela sua passagem. Ao detectar a permissão livre, a estação mudará o pacote de controle para permissão ocupada, e fará em seguida a transmissão desejada. Ao término da transmissão, a estação emitirá um pacote de controle, indicando permissão livre, e ficará esperando pela volta completa do pacote de controle e pacote de informação, emitidos anteriormente, para retirá-los do anel. No momento de retirar o pacote de controle do anel, a estação origem poderá verificar os bits de resposta preenchidos pela estação destino e tomar as providências necessárias. Deverão estar previstos no anel, mecanismos de tratamento de erros e de recuperação de permissão.

CAPÍTULO IV

SOFTWARE E ESTRUTURAÇÃO DE SISTEMAS DISTRIBUÍDOS

O aparecimento de sistemas de processamento paralelo, envolvendo sistemas multiprogramados, multiprocessadores, redes de computadores, e sistemas distribuídos, exerceu forte influência na evolução do software de sistemas, principalmente no projeto de sistemas operacionais.

Inicialmente, os projetos de sistemas operacionais para sistemas de computação mais simples eram implementados com o uso de linguagem de montagem ("assembly"), o que permitia aos seus projetistas tratarem da estrutura lógica do sistema, e também dos detalhes de máquina. Com o crescimento do sistema operacional, em relação ao tamanho e complexidade, principalmente para sistemas de processamento paralelo, surgiu a necessidade de dividí-lo racionalmente (POPEK e KLINE [130]). Os esforços foram então dirigidos para o desenvolvimento de uma base, num nível próximo da máquina, que oferecesse as funções básicas de um sistema, de maneira altamente confiável. Assim, o sistema operacional passou a ser estruturado por camadas, ou seja, em módulos hierarquicamente dispostos, onde o módulo básico implementava as funções básicas do sistema e cuidava dos detalhes de máquina, e os módulos superiores tratavam da manipulação da estrutura lógica do sistema. Essa técnica de desenvolvimento de sistema operacional, utilizando o módulo básico denominado de "nucleus", "kernel", ou mesmo "monitor", foi adotada em vários projetos de sistema operacional a partir do início da década de 70. Desde então, tal técnica foi aperfeiçoada, tanto para sistemas simples com uma única UCP quanto para sistemas complexos compostos por múltiplas UCPs ou múltiplos computadores.

Os sistemas complexos, envolvendo processamento fisicamente paralelo, particularmente aqueles com arquiteturas convencionais, abrangendo multiprocessadores, redes de computadores, e sistemas distribuídos, possuem características de hardware e de

software bem definidas (ENSLOW JR. [50], TANEMBAUN [178]):

- um sistema multiprocessador é formado por múltiplas UCPs interligadas através de uma memória compartilhada, e apresenta um sistema operacional integrado que gerencia os recursos do sistema como um todo;

- uma rede de computadores, por sua vez, é formada por múltiplos computadores autônomos interligados através de um subsistema de comunicação, envolvendo canais de comunicação. Quanto ao software, a rede apresenta um sistema operacional que implementa a infraestrutura necessária para a comunicação entre os vários computadores, respeitando e mantendo o sistema operacional de cada um;

- um sistema distribuído possui um hardware formado por uma rede de computadores, e apresenta um sistema operacional integrado que gerencia os recursos do sistema como um todo.

Apesar das diferenças de estrutura e de implementação, tanto os multiprocessadores quanto os sistemas distribuídos apresentam-se ao usuário como um sistema unitário, enquanto que a rede de computadores deixa transparecer ao usuário a existência explícita de seus diversos computadores e recursos existentes. Por serem apresentados de forma integrada e possuirem algumas semelhanças, os sistemas operacionais para multiprocessadores e os sistemas operacionais distribuídos serão analisados detalhadamente neste trabalho.

IV.1 - SOFTWARE DISTRIBUÍDO

Dos quatro elementos de um sistema que podem estar distribuídos: hardware; dados; programas; e controle; os três últimos referem-se ao software. A distribuição dos dados levou ao desenvolvimento de uma área bastante extensa, envolvendo sistemas de arquivos distribuídos (GIEN [59], HAC [63], STURGIS et alii [176]), e sistemas de banco de dados distribuído (DAVEN

PORT [40,41], KATZAN JR. [81], BERNSTEIN e GOODMAN [15]), que não serão abordados neste trabalho. A seguir, serão considerados os aspectos de distribuição de programas e de controle, cuja influência no desenvolvimento de sistemas operacionais tem sido determinante.

Para que a distribuição de programas (algorítmos) seja tratada, é necessário que se tenha o seguinte entendimento:

- existem programas sequenciais, cujas instruções são executadas uma por vez, e programas paralelos ou concorrentes, cujas instruções, em conjuntos bem definidos, podem ser executadas ao mesmo tempo;

- um programa é dito centralizado, se o processador (ambiente) puder executar potencialmente qualquer uma de suas instruções;

- um programa é dito distribuído, se estiver espalhado por vários ambientes, ou seja, se alguns processadores (ambientes) não puderem executar alguns passos do programa, pelo fato deles pertencerem a outros ambientes;

- qualquer programa sequencial ou paralelo pode ser um programa distribuído.

Diante disso, pode-se dizer que qualquer programa que for executado numa arquitetura, onde os processadores possam executar qualquer instrução desse programa, será um programa centralizado; é o caso do sistema multiprocessador composto por uma única memória compartilhada e vários processadores.

Por outro lado, um programa será distribuído, se puder ser estruturado em partes, formadas por grupos de instrução e conhecidas como componentes do programa, colocadas de forma que cada parte só possa ser executada pelo processador a ela associado. Os componentes do programa separados pelo subsistema de comunicação são conhecidos como remotos uns aos outros. A troca de informações entre componentes remotos do programa deverá ocorrer por transferência de mensagens, através do meio de comunicação. Como exemplo de programa distribuído pode-se citar o programa sequencial formado por um programa principal e sua subrotina, loca

lizados de forma a serem remotos um ao outro.

A distribuição do controle, por sua vez, está relacionada com o processamento do programa, ou seja, com o estado da computação, significando a contrapartida dinâmica do programa.

Existirá um controle centralizado num sistema, quando o progresso completo da execução do programa, em qualquer instante, ficar sob controle de um único processador. É o caso de um supervisor de multiprocessador, cuja execução só ocorra num processador específico. É o caso também de um programa distribuído, cujas partes são trazidas a um único processador para execução.

Se o progresso da execução do programa passar pelo controle de mais do que um processador, durante o processamento, o controle será distribuído. É o caso da chamada remota de procedimento, onde o controle é transferido temporariamente, retornando em seguida para o processador inicial. É o caso também de processamento em cadeia com transferência de resultados intermediários, onde o controle é transferido definitivamente de um processador a outro.

IV.2 - ESTRUTURAÇÃO DE SISTEMAS DISTRIBUÍDOS

A utilização de minicomputadores e microcomputadores em redes de computadores imprimiu um grande desenvolvimento na área de sistemas distribuídos, pelo fato de viabilizarem a existência de sistemas com grande número de elementos autônomos cooperativos. Então, foi possível experimentar diversas estruturas, onde os programas do sistema e do usuário pudessem ser divididos em partes, e executados em máquinas diferentes. As variações dessas estruturas, do ponto de vista físico e lógico, serão analisadas a seguir.

IV.2.1 - ESTRUTURAÇÃO BASEADA NA DISTRIBUIÇÃO FÍSICA

Existem vários modelos de sistemas distribuídos, baseados na distribuição física dos seus elementos, envolvendo desde

grandes redes, cujas máquinas frouxamente conectadas comunicam-se raramente, até sistemas multiprocessadores, cujos elementos fortemente conectados chegam a interagir intensamente.

Dentre os vários modelos baseados na distribuição física, encontram-se os seguintes: modelo hierárquico; modelo do "cache" de UCP; modelo usuário-servidor; e modelo do conjunto de processadores (TANENBAUM [178]).

No modelo hierárquico, os computadores são dispostos em uma rede sob a forma de árvore, de maneira que quanto mais próximos estiverem da raiz, mais potentes deverão ser. O computador da raiz tratará, de forma geral, do sistema como um todo, enquanto que os computadores distantes da raiz tratarão de tarefas específicas e especializadas, resumindo-as para os computadores situados em níveis que vão aproximando-se da raiz. Esse tipo de estrutura é comumente utilizada, quando os problemas são organizados naturalmente de forma hierárquica, como ocorre em muitas empresas.

O modelo do "cache" de UCP consiste na utilização de computadores de menor porte como elementos que interligam terminais com uma grande UCP. Nesse caso, uma parte do processamento será executada no computador ligado ao terminal, e a outra parte, no computador central. Analogamente ao sistema "cache" de memória que funciona com dois níveis de memória, envolvendo uma memória lenta e uma memória rápida, o "cache" de UCP funciona com dois níveis de processamento: um lento e outro rápido. Os programas deverão estar disponíveis na máquina grande e em uma máquina pequena. A decisão de executar o programa em uma ou outra máquina, levará em conta fatores das duas, tais como: adequabilidade; taxa de saída; custos relativos; e carga de trabalho.

O modelo usuário-servidor surgiu na medida em que os pequenos computadores, crescendo em potência e tendo seus preços reduzidos, diminuiram gradativamente a importância do computador central do modelo do "cache" de UCP. Por outro lado, a possibilidade de comunicação entre os usuários e o compartilhamento de recursos caros e escassos determinaram o aparecimento de sistemas, formados por vários computadores usuários e vários computadores servidores interligados através de uma subrede de comuni

cação. Um computador usuário apresenta-se tipicamente como um computador pessoal, contendo um terminal de vídeo e eventualmente um pequeno sistema de disco. Esse computador tem a função de executar programas do usuário, e de transferir requisições a outros computadores através da rede. Um computador servidor, ligado a um periférico ou a um conjunto de periféricos, ou mesmo, funcionando isoladamente, tem a função de atender requisições dos usuários, mediante o estabelecimento de serviços bem definidos a nível de sistema. Tais serviços envolvem, por exemplo: execução de entrada e saída; manipulação de arquivos e de banco de dados; tratamento de temporização; realização de autenticação; etc. Esta estrutura, embora possa satisfazer as necessidades do usuário, apresenta o problema de ser pouco eficiente quanto a exploração de todo seu potencial, pois principalmente os computadores usuários ficarão sub-utilizados, quando estiverem atendendo entrada e edição de programas.

O modelo do conjunto de processadores ("pool") procura contornar o problema da ineficiência do modelo usuário-servidor, eliminando os computadores usuários e fazendo com que os terminais sejam ligados à rede, de forma direta ou através de concentradores de terminais. Para compensar a eliminação dos computadores usuários, esta estrutura prevê a existência de um conjunto de processadores de execução ("pool"), encarregados de executar dinamicamente, programas de usuário e funções do sistema, de acordo com a necessidade. Os computadores servidores continuarão existindo e poderão atender aos usuários, na fase de preparação de programas. O aumento de eficiência ocorrerá em função de um aumento na complexidade.

IV.2.2 - ESTRUTURAÇÃO LÓGICA

A estruturação lógica de sistemas distribuídos, principalmente de sistemas operacionais distribuídos, utiliza basicamente o modelo de processos ou o modelo de objetos (BACON [11], TANEMBAUM [178]).

IV.2.2.1 - MODELO DE PROCESSOS

O modelo de processos baseia-se na utilização de elementos ativos (processos) para a estruturação de todo o sistema. Esses processos podem encapsular tanto elementos ativos por natureza, consistindo dos programas referentes às atividades do sistema e do usuário, quanto elementos naturalmente passivos correspondendo aos recursos e suas respectivas operações, confinados em gerenciadores de recursos. Os processos referentes às atividades ativas que, de alguma maneira, cooperam e competem entre si, são denominados processos clientes. Os processos encarregados de atender requisições dos processos clientes, no sentido de utilizar recursos e executar serviços tradicionais do sistema operacional, são conhecidos como processos servidores. A comunicação entre os processos, implementada pelo sistema operacional, pode ocorrer de duas formas, ou seja, através de chamada remota de procedimento, e através de troca explícita de mensagem.

A chamada remota de procedimento (NELSON [121], LISKOV [104]), embora ocorrendo num ambiente, onde os processos e procedimentos encontram-se distribuídos, tem a mesma semântica que uma chamada de procedimento local: o processo emissor da chamada de procedimento ficará bloqueado até que o referido procedimento (receptor da chamada) seja executado, e seus parâmetros de retorno sejam passados ao processo emissor. Se o processo emissor da chamada e processo receptor (procedimento) não compartilharem nenhuma memória comum, a passagem de parâmetros na chamada e no retorno deverá ser feita por valor, não podendo ser utilizado o esquema de passagem de parâmetros por referência (endereço). Este tipo de comunicação entre processos é bastante dependente da linguagem usada para a implementação do sistema, devendo satisfazer suas restrições com relação à chamada de procedimento e à passagem de parâmetros. Existem muitos trabalhos relacionados com chamada remota de procedimento, dentre eles aqueles citados em NELSON [121], BIRRELL [17], BIRRELL e NELSON [18], SHRIVASTAVA e PANZIERI [158], e CARPENTER e CAILLIAU [33].

A troca explícita de mensagem entre processos que se comunicam, conforme LAUER e NEEDHAM [98] e GENTLEMAN [58], ba

seia-se na existência de um padrão de comunicação previamente estabelecido. Este padrão faz com que uma cadeia de bits ou um conjunto de palavras, constituindo a mensagem, tenha sentido para os vários processos do sistema. A mensagem, além de embutir a informação útil que será usada diretamente pelos processos, leva também informações complementares essenciais para sua movimentação pelo sistema. Essas informações complementares caracterizam a mensagem, indicam sua origem e seu destino, e auxiliam na detecção e correção de erros. A transferência de mensagens de um processo para outro é feita por operações primitivas, implementadas ao nível do sistema operacional e do núcleo, consistindo basicamente de operações de emissão e de recepção de mensagens. Apesar de apresentar muitas variações, dependendo das características do sistema e do tipo de implementação que se deseja realizar, as operações de comunicação têm o seguinte significado genérico: a emissão ("send") resume-se na atividade de encaminhar, ou iniciar o encaminhamento de mensagens de um processo a outro ou a vários processos; a operação de recepção ("receive") consiste na atividade de receber, ou procurar receber mensagens de um processo específico ou de qualquer processo. Das várias questões a serem tratadas, quando se utiliza troca de mensagens, pode-se citar as seguintes: formas de endereçamento e acesso aos processos; formas de transmissão das mensagens; tipos de comunicação entre processos; tipos de primitivas; utilização de "buffers"; etc. Esse esquema de comunicação entre processos é mais flexível do que a chamada remota de procedimento, uma vez que as restrições estão relacionadas com a existência de ligações implícitas ou explícitas entre os processos, e com a interpretação da mensagem. A comunicação por troca de mensagens permite que cada processo seja independente dos outros, podendo ser escrito com linguagens diferentes em cada caso. As restrições sobre a comunicação entre processos podem ser mantidas dinamicamente. Algumas estruturas baseadas em troca de mensagens foram desenvolvidas para facilitar o gerenciamento de recursos como, por exemplo, o gerente de recursos (JAMMEL e STIEGLER [72]), e o serializador (HEWITT e ATKINSON [65]).

IV.2.2.2 - MODELO DE OBJETOS

O modelo de objetos baseia-se no encapsulamento das várias partes de um sistema em elementos denominados objetos (JONES [74], LAMPSON e STURGIS [97]). Os objetos dividem-se em dois grupos:

- aqueles que encapsulam processos que, depois de iniciados, constituem a parte ativa do sistema, sendo chamados simplesmente de processos;

- aqueles que encapsulam recursos e facilidades, cuja função resume-se em atender passivamente as requisições provenientes dos processos, sendo chamados simplesmente de objetos.

Os objetos são estruturados de forma a apresentarem um conjunto de operações responsáveis pelo seu comportamento. Para conseguir acessar um objeto, um processo deve ter capacidade para fazê-lo, usando o conhecimento do nome do objeto e a posse da autorização para acessar algumas ou todas as suas operações. A permissão para executar uma função específica sobre um objeto é definida como um direito. O sistema deverá manter, para cada processo, uma lista de capacidades, indicando os objetos e as operações, aos quais o processo poderá ter acesso. A alteração dessas listas de capacidade deverá ser feita de maneira segura para não colocar em risco a integridade do sistema, sempre que nele se fizer necessária uma adaptação ou alteração. A estrutura de monitor (HOARE [66], BRINCH HANSEN [21,22,24]), inspirada no mecanismo de classe da linguagem Simula 67, conforme KEEDY [82], é uma aplicação do conceito de objeto e abstração de dados. O monitor foi incorporado em várias linguagens para programação concorrente, como: Pascal Concorrente (BRINCH HANSEN [23]); CSP/K (HOLT et alii [69]); Concurrent Euclid (HOLT [68]); etc. O conceito de objeto também foi utilizado no desenvolvimento de vários projetos de núcleo e de sistemas operacionais, tais como: Star OS (JONES et alii [76]); HYDRA (WULF et alii [204]); Cal System (LAMPSON e STURGIS [97]); etc.

IV.3 - LINGUAGENS PARA PROGRAMAÇÃO DE SISTEMAS

Linguagens e sistemas operacionais têm sido desenvolvidos de forma independente ao longo do tempo, apresentando pouca coisa em comum. No entanto, a aproximação dessas duas áreas aconteceu, quando as novas linguagens de alto nível passaram a incorporar facilidades, antes encontradas somente nos sistemas operacionais como, por exemplo, mecanismos de comunicação e sincronização entre processos. Além disso, o uso de linguagens de alto nível veio contribuir para melhores condições de verificação de programa, devido à existência de comandos estruturados e de tipos abstratos de dados, e de outras características.

A utilização da mesma linguagen de alto nível, ou de subconjuntos dela, por parte do usuário para programar suas aplicações, e por parte do programador de sistemas para programar o sistema operacional, pode trazer várias vantagens (JONES [73]):

- o usuário encontra menor dificuldade para assimilar a linguagem e o sistema, uma vez que ambos envolvem um único conjunto de conceitos básicos;

- certas características do sistema operacional ficam visíveis na linguagem, independentemente se elas são tratadas estaticamente pelo compilador, ou dinamicamente pelo sistema operacional;

- os programas do usuário podem ser visualizados como uma extensão do sistema operacional.

As linguagens utilizadas para programar sistemas multiprocessadores e sistemas distribuídos enquadram-se em uma das seguintes categorias (ANDREWS [7]):

a) linguagens sequenciais ampliadas com rotinas escritas em "assembly" para tratamento do hardware;

b) linguagens concorrentes baseadas em variáveis compartilhadas;

c) linguagens concorrentes baseadas em troca de mensagens explícita ou implícita.

Embora as linguagens sequenciais, ampliadas com rotinas escritas em "assembly", existam há mais tempo e tenham sido muito utilizadas, as linguagens concorrentes estão cada vez mais ampliando o seu espaço de atuação.

IV.3.1 - CLASSES DE LINGUAGENS PARA PROGRAMAÇÃO CONCORRENTE

Na análise de linguagens concorrentes, um dos aspectos de grande importância que serve inclusive como elemento de classificação dessas linguagens é a forma de comunicação entre processos. Por esse motivo, antes de abordar as características e requisitos de linguagens apropriadas para a programação de sistemas, convém enfocar esse aspecto específico das linguagens concorrentes.

A interação entre processos utiliza-se normalmente de uma dentre duas formas básicas de comunicação: através de chamadas de procedimentos, ou através de troca de mensagens. A partir de uma composição dessas duas formas de comunicação com as duas maneiras de sincronização (síncrona e assíncrona), é possível enquadrar algumas linguagens dentro de um diagrama ortogonal, envolvendo as duas formas de comunicação e as duas formas de sincronização (STAUNSTRUP [173]). Incluindo-se os mecanismos de comunicação e sincronização mais utilizados no referido diagrama, obtém-se a figura (IV.1), que apresenta a relação entre algumas linguagens e mecanismos de comunicação e sincronização, tendo como fundo as possibilidades de comunicação e sincronização.

Os quadrantes da direita do diagrama da figura (IV.1) (troca de mensagens) envolvem a comunicação entre processos, através do uso de primitivas do tipo *envia* e *recebe* num ambiente, onde os objetos do sistema são colocados em processos. Os processos encarregados de executar as operações necessárias sobre cada objeto são denominados gerentes de objeto. Uma operação sobre um objeto é realizada, através da transferência de mensagens do processo requisitante para o gerente de objeto. Eventualmente, no caso do gerente de objeto precisar emitir respostas, haverá transferência de mensagens no sentido inverso da requisição.

No quadrante superior direito (troca de mensagem - assíncrona), encontram-se as primitivas, cuja execução independe da

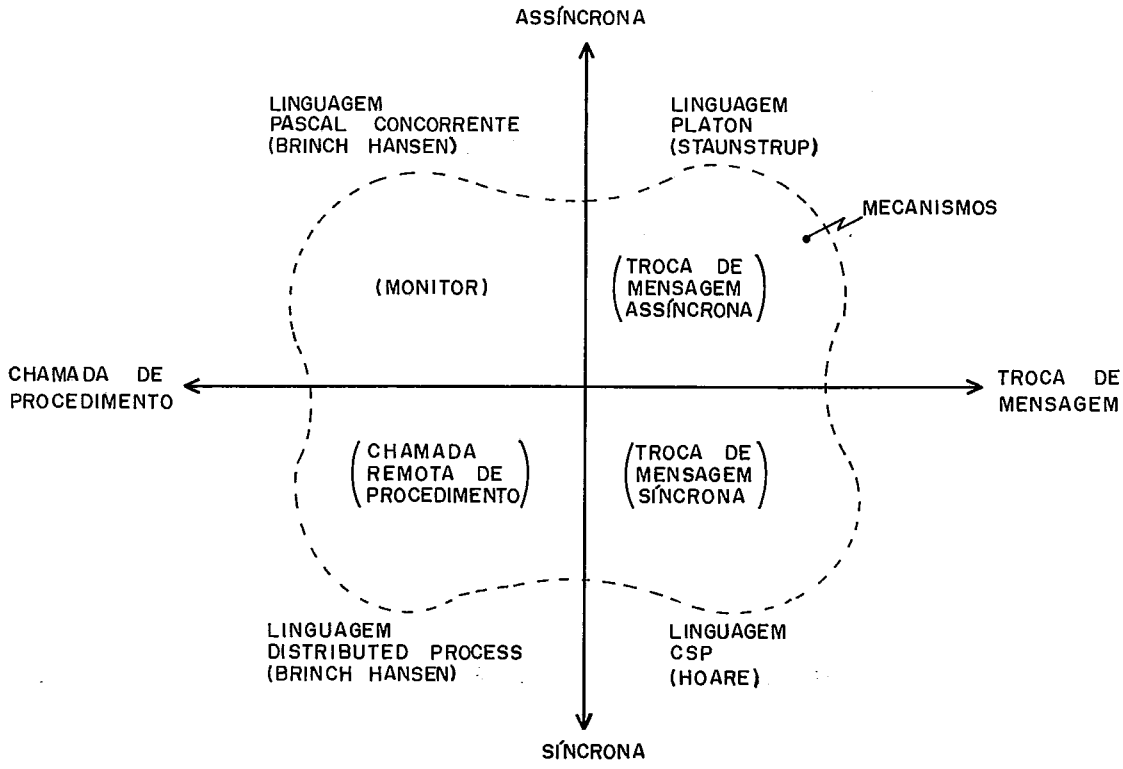


FIGURA IV . I - DIAGRAMA ORTOGONAL SOBRE COMUNICAÇÃO EM PROGRAMAÇÃO CONCORRENTE

situação dos processos emissores e receptores. A execução dessas primitivas acontecerá de forma não bloqueada, ou seja, os processos que contiverem as primitivas terão continuidade, mesmo que não estejam prontos (sincronizados) para a comunicação.

No quadrante inferior direito (troca de mensagem - síncrona), situam-se as primitivas, cuja execução dependerá da situação dos processos emissores e receptores. Estes processos serão bloqueados, quando os processos parceiros não estiverem prontos para a comunicação.

Os quadrantes esquerdos (chamada de procedimento) envolvem uma estrutura de comunicação de nível mais alto baseada na chamada de procedimentos, onde um processo, ao chamar um procedimento, ficará bloqueado no ponto da chamada até que o procedimento termine sua execução; em seguida, o processo chamador será liberado para continuar.

No quadrante superior esquerdo (chamada de procedimento-

assíncrona), onde situa-se o mecanismo conhecido por monitor (HOARE [66]), os procedimentos fazem parte de uma estrutura passiva. Quando uma instrução de um processo, correspondendo à chamada de um procedimento remoto, for executada, o processo será suspenso e a chamada de procedimento será encaminhada, mesmo que o procedimento não possa ser executado imediatamente; isto permite que a chamada seja caracterizada como assíncrona. A chamada ficará pendente numa fila situada junto à estrutura, onde estiver o procedimento, e a sua execução será realizada tão logo seja possível.

No quadrante inferior esquerdo (chamada de procedimento - síncrona), onde situa-se a chamada remota de procedimento (NELSON [121], LISKOV [104]), os procedimentos estão associados a processos que gerenciam as suas chamadas. É necessário que haja uma sincronização entre o processo chamador do procedimento, no ponto da chamada, e o processo gerenciador dos procedimentos, no ponto de recepção da chamada, para que a execução do procedimento seja iniciada. O princípio básico de uma chamada remota de procedimento está no fato de uma *chamada de procedimento* com seus parâmetros ser interceptada, e, em vez de ser executada no mesmo computador, ser transmitida para execução em outro computador. Enquanto isso, o programa que tiver chamado o procedimento permanecerá bloqueado. Quando o procedimento terminar no computador remoto, o *retorno* com seus parâmetros será interceptado e remetido para o computador que originou a chamada. Aí então, o programa que fez a chamada será desbloqueado, e continuará como se a chamada tivesse sido executada localmente.

Outra maneira de considerar as linguagens concorrentes, em relação aos mecanismos de interação entre processos, consiste em enquadrá-las em uma das três classes seguintes: linguagens orientadas a procedimentos; linguagens orientadas a mensagens; e linguagens orientadas a operações (ANDREWS e SCHNEIDER [9]). A figura (IV.2) mostra a relação entre técnicas de sincronização e essas três classes de linguagens.

As linguagens orientadas a procedimentos são aquelas que incorporam chamada assíncrona de procedimento. Essas linguagens tratam de objetos ativos (processos) e de objetos passivos compartilhados (módulos, monitores, etc.). A interação entre

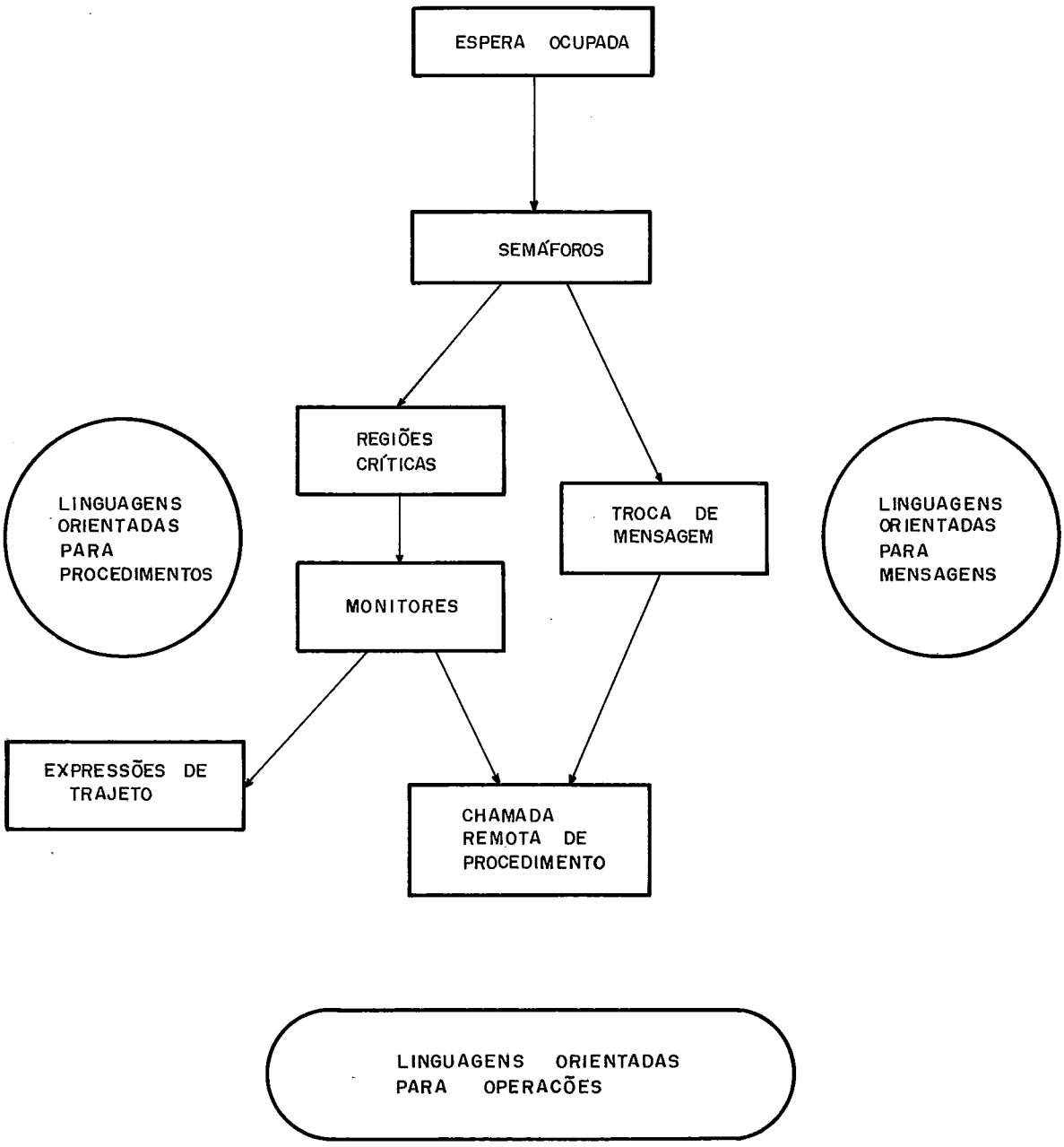


FIGURA IV . 2 - RELAÇÃO ENTRE TÉCNICAS DE SINCRONIZAÇÃO E CLASSE DE LINGUAGEM.

processos baseia-se em variáveis compartilhadas que representam os objetos passivos. A manipulação dos objetos pelos processos dá-se através de chamada de procedimento. Como exemplos desse tipo de linguagem pode-se citar: Pascal Concorrente (BRINCH HANSEN [23]); Módula (WIRTH [200]); Mesa (LAMPSON e REDELL [96]); Edison (BRINCH HANSEN [28,29]); CSP/K (HOLT et alii [69]); Concurrent Euclid (HOLT [68]); etc.

As linguagens orientadas a mensagens são aquelas que tratam da troca explícita de mensagens. A interação entre os processos que constituem os elementos do sistema baseia-se simplesmente na troca de mensagens entre eles. As linguagens CSP (HOARE [67]) e PLITS (FELDMAN [53]) são exemplos dessa classe.

As linguagens orientadas a operação combinam alguns aspectos das duas classes anteriores. Cada objeto está associado a um processo gerente, e as operações são realizadas sobre o objeto através da execução de um procedimento. A interação entre processos ocorre pela troca de mensagem, precedendo e sucedendo a execução de um procedimento acessado pelo processo gerente respectivo. As linguagens DP (BRINCH HANSEN [26]), Star Mod (COOK [37]), ADA (ICHBIAH et alii [71], WEGNER [197]), e SR (ANDREWS [6,8]) são exemplos desta classe.

A utilização dessas linguagens está bastante relacionada com o tipo de ambiente, onde elas venham a atuar. As linguagens orientadas a procedimento são mais adequadas a sistemas monoprocessadores ou a multiprocessadores. As linguagens orientadas a mensagem são mais apropriadas para sistemas distribuídos, embora apresentem boa adequação para sistemas com memória compartilhada. As linguagens orientadas a operação incorporam as vantagens das duas classes anteriores; elas poderão ser implementadas como linguagens orientadas a procedimento, quando existir memória compartilhada, ou poderão usar troca de mensagem, no caso de sistemas distribuídos. De qualquer maneira, as três classes de linguagem podem ser adaptadas aos vários tipos de ambiente, apresentando, ou não, uma aderência natural ao ambiente, e implicando em sobrecargas diferentes de execução, dependendo do estilo de programação.

As linguagens orientadas a procedimentos e orientadas a mensagens foram tratadas por LAUER e NEEDHAM [98] e REID [141], enfocando a questão da dualidade desses dois tipos de linguagem.

O terceiro tipo de linguagem, pelo fato de ser mais recente, não foi abordado nessas discussões. Num nível abstrato, pode-se considerar que esses três tipos de linguagens são intercambiáveis.

IV.3.2 - CARACTERÍSTICAS E REQUISITOS DE LINGUAGENS PARA PROGRAMAÇÃO DE SISTEMAS

Do ponto de vista do programador, um sistema distribuído será um sistema tolerante a falhas elaborado com módulos remotos de programa. A comunicação entre esses módulos dar-se-á através de troca de mensagens, podendo gastar tempos diferentes em cada tentativa.

Uma linguagem para programação de sistemas num ambiente distribuído deve ser simples, portátil, e adequada para a elaboração de programas distribuídos e de programas de aplicação em tempo real. Para isso, a linguagem deve possuir vários requisitos. LI e LIU [101,106] restringiram sua análise aos aspectos de linguagem, voltados ao desenvolvimento de sistemas distribuídos, consistindo dos seguintes ítens: concorrência; comunicação; sincronização; dependência do tempo; e tolerância a falhas. Outros autores, como PARDO [123], APPELBE [10], TISATO e ZICARI [186], STOTTS JR. [175], e SEEGMÜLLER [149], abordaram listas ampliadas de características de linguagens para programação de sistemas, envolvendo algumas ou todas as seguintes características:

- *método de comunicação*: corresponde à maneira pela qual os processos trocam informações; os métodos normalmente utilizados envolvem troca de mensagens, variáveis globais (variáveis compartilhadas desprotegidas), dados compartilhados protegidos por módulos ou monitores, e chamada remota de procedimento;

- *método de sincronização*: corresponde à forma de imposição de restrições no andamento dos processos; dentre os vários métodos, pode-se citar aqueles que utilizam sinais, troca de mensagem síncrona, "buffers", expressões de trajeto, eventos, filas de condição, regiões com guarda, "rendez-vous" estendido, chamada remota de procedimento, e transferência explícita de

co-rotina;

- *uso de variáveis globais* (variáveis compartilhadas desprotegidas): é a forma mais antiga de comunicação de dados, e possivelmente de sincronização entre processos. Essas variáveis são excluídas de algumas linguagens, por serem consideradas inconvenientes;

- *uso de "buffers"/comprimento definido pelo usuário*: os "buffers" de mensagem, possivelmente com comprimento infinito, podem existir em uma linguagem como um tipo de dado pré-definido. Os "buffers" podem ser criados como variáveis de programa com comprimento especificado pelo usuário;

- *criação de processos*: os processos concorrentes são criados estaticamente no início da execução do programa, ou são criados dinamicamente, por ocasião de sua chamada durante a execução. Processos estáticos continuam existindo, mesmo depois de não terem razão de ser, enquanto que processos dinâmicos geralmente desaparecem depois de terminados;

- *rede de interconexão de processos*: o esquema de interconexão estática de processos é aquele, onde as ligações de comunicação entre os processos não mudam durante a execução, ou, se houver criação e destruição dinâmicas de processos, a colocação e retirada de ligações estará prevista em tempo de compilação. Interconexão dinâmica indica que as ligações entre processos podem alterar-se durante a execução;

- *execução indeterminada/expressão explícita para a indeterminação*: indica a capacidade de uma linguagem em expressar possíveis execuções indeterminadas, significando que, para várias execuções sobre os mesmos dados de entrada, pode-se obter conjuntos diferentes de dados de saída. Algumas linguagens possuem declarações especiais para expressar explicitamente execução indeterminada, enquanto em outras a indeterminação acontece através de algum escalador de processos

ou de mecanismo de fila, os quais são inacessíveis ao programador:

- *compilação em separado*: corresponde à possibilidade de compilar um módulo de código, mesmo sem o conhecimento do código do restante do programa. Geralmente, necessita-se de alguma especificação dos outros módulos para poder-se fazer verificação de tipo envolvendo os módulos de interface;

- *suporte de tempo real*: significa que a linguagem possui características que permitem ao programador controlar explicitamente a ordem de execução dos processos, manipular saídas por tempo ("time-out") na fase de comunicação entre processos, comunicar-se diretamente com dispositivos de hardware existentes no ambiente, etc. Tais características tornam a programação de aplicações em tempo real mais fácil e confiável;

- *facilidade de abstração*: significa que a linguagem pode manipular ítens como monitores, módulos, pacotes, tipos abstratos de dados, etc.;

- *tratamento de exceções*: corresponde à existência de facilidades para ações definidas pelo programador, a serem tomadas, quando ocorrerem condições excepcionais durante o tempo de execução;

- *suporte para verificação de programas*: a linguagem pode possuir características que ajudem os programadores ou verificadores automáticos, a determinar a correção lógica do programa. Essas características podem estar relacionadas com construções sintáticas, incluindo especificação formal de processos e suportes semânticos como, por exemplo, convenções de chamada de processos e de monitores que restringem as instâncias de comunicação a estruturas hierárquicas não recursivas.

Uma análise compacta de várias linguagens para programação concorrente, relativas às características listadas ante

riormente, consta no quadro (IV.1). Esse quadro agrupa na sua parte esquerda algumas linguagens orientadas a procedimento, enquanto na parte direita estão linguagens orientadas a mensagens; algumas linguagens orientadas a operações encontram-se na parte central do quadro. A linguagem Módula-2 (WIRTH [201,202]), por utilizar o mecanismo de interação entre processos baseado em co-rotinas, não foi classificada dentro desses três tipos de linguagens, ficando na coluna referente a outras linguagens.

Todas as linguagens consideradas no quadro são linguagens de alto nível que, além de apresentarem as vantagens inerentes a esse tipo de linguagem, possuem características adicionais. Estas características recomendam em grau variado o emprego dessas linguagens em programação de sistemas.

Uma linguagen de programação de sistemas deve ser eficiente e confiável. Nesse sentido, o código objeto gerado pelo compilador deve ser tão eficiente quanto o código produzido por um programador experiente, usando a linguagem "assembly"; os erros de desenvolvimento, usando a linguagem de alto nível, devem ser detectados de preferência na fase de compilação.

Além disso, uma linguagem deve ser mais completa possível, de forma a oferecer todos os elementos necessários para que a implementação do sistema utilize somente os recursos da linguagem. Nem todas as linguagens são projetadas com esse objetivo, devido ao fato de muitas vezes conseguir-se que a linguagem seja completa às custas de características como simplicidade e confiabilidade, e até às custas de sobrecarga do software e do hardware que precisam dar suporte às construções de linguagem de alto nível. Muitas vezes, é mais interessante ter-se linguagens incompletas, que são complementadas com mecanismos fornecidos por rotinas de biblioteca suportadas pelo sistema operacional. Essa alternativa apresenta o inconveniente de reduzir a portabilidade, uma vez que a biblioteca, que frequentemente incorpora características locais, precisa ser transportada junto com os programas.

De uma análise do quadro mencionado pode-se concluir que as linguagens Ada, Mesa, e também PLITS com alguma restrição são linguagens completas, enquanto que as restantes podem ser qualificadas de incompletas, apesar de apresentarem inúmeras vantagens para aplicações em projetos de sistemas.

TIPOS	LINGUAGENS ORIENTADAS A PROCEDIMENTOS				LINGUAGENS ORIENTADAS A OPERAÇÕES		LINGUAGENS ORIENTADAS A MENSAGENS		OUTRAS LINGUAGENS
EXEMPLOS / CARACTERÍSTICAS	PASCAL CONCORRENTE	MÓDULA	EDISON	MESA	DP	ADA	CSP	PLITS	MÓDULA-2
MÉTODO DE COMUNICAÇÃO	monitor	monitor	monitor	monitor	procedimento	"rendez-vous" estendido	mensagem	mensagem	módulo
MÉTODO DE SINCRONIZAÇÃO	fila ("queue")	sinais	condição	condição	regiões c/ guarda	"rendez-vous" estendido	troca de mensagem síncrona	"buffer"	transferência
USO DE VARIÁVEIS GLOBAIS	não	sim	sim	sim	não	sim	não	não	sim
USO DE "BUFFERS" / COMPRIMENTO DEFINIDO P/ USUÁRIO	não/não	não/não	não/não	não/não	não/não	não/não	não/não	sim/não	não/não
EXECUÇÃO INDETERM. / EXPRESSÃO EXPLÍCITA P/INDET.	sim/não	sim/não	sim/não	sim/não	sim/sim	sim/sim	sim/sim	sim/não	sim/não
SUPORTE DE TEMPO REAL	sim	sim	sim	sim	sim	sim	não	não	sim
TRATAMENTO DE EXCEÇÕES	não	não	não	sim	não	sim	não	sim	não
CRIAÇÃO DE PROCESSOS	estática	dinâmica	dinâmica	dinâmica	estática	dinâmica	dinâmica	dinâmica	dinâmica
INTERLIGAÇÃO DE PROCESSOS	dinâmica	estática	estática	dinâmica	estática	dinâmica	estática	dinâmica	dinâmica
FACILIDADE DE ABSTRAÇÃO	pouca	pouca	pouca	média	pouca	muita	pouca	pouca	média
COMPILAÇÃO EM SEPARADO	não	não	não	sim	não	sim	não	sim	sim
SUPORTE PARA VERIFICAÇÃO DE PROGRAMA	médio	pouco	pouco	pouco	pouco	pouco	pouco	muito	pouco

QUADRO IV.1 - CARACTERÍSTICAS DE ALGUMAS LINGUAGENS PARA PROGRAMAÇÃO CONCORRENTE

CAPÍTULO V

SISTEMAS OPERACIONAIS DISTRIBUÍDOS

Os sistemas operacionais distribuídos são grandes e complexos por natureza, uma vez que atuam em ambientes formados por múltiplos computadores ou múltiplos processadores. Desta maneira, a estruturação desses sistemas operacionais constitui-se numa questão fundamental dentro do contexto da organização de software de sistemas distribuídos (FINKEL [54]). Além disso, constata-se que a grande maioria dos sistemas operacionais distribuídos discutidos na literatura apresentam uma estruturação, envolvendo alguma forma de núcleo.

O núcleo é a base sobre a qual se constrói o sistema operacional distribuído. Sua função lógica consiste em transformar o hardware e o software de baixo nível do sistema em uma máquina virtual que ofereça as facilidades necessárias ao sistema operacional. Devido ao fato do núcleo funcionar como uma extensão da máquina, ele constitui-se num elemento intensamente utilizado que, se não for bem projetado, poderá atuar como um gargalo do sistema e comprometer fatores tais como desempenho e segurança. Para evitar-se tais problemas, as questões de projeto de núcleo devem ser cuidadosamente estudadas, e os conflitos de implementação resolvidos para obter-se a melhor solução.

V.1 - NÚCLEO

O núcleo de um sistema operacional distribuído, ou de uma linguagem usada para programá-lo, é formado pelo conjunto de vários núcleos individuais, conforme a figura (V.1). Esses núcleos associados às várias unidades de computadores ou processadores, que interligados compõem o sistema distribuído, podem ser iguais ou diferentes. Os núcleos individuais devem atuar,

cada um na sua esfera de ação, e cooperar entre si no sentido de gerar um ambiente uniforme capaz de fornecer o suporte necessário a sistemas operacionais.

V.1.1 - FORMULAÇÃO DO NÚCLEO

O projeto de um núcleo deve tratar do estabelecimento das funções que deverão ser incorporadas ao núcleo, e da forma de sua estruturação (LORIN e DEITEL [108]). Os núcleos podem variar em natureza e tamanho, dependendo do tipo de sistema operacional ao qual eles dão suporte, servindo de base, tanto para sistemas de grande porte apropriados para uso geral quanto para sistemas especializados de pequeno porte. Além disso, um núcleo pode ser mínimo, contendo somente as informações necessárias pa

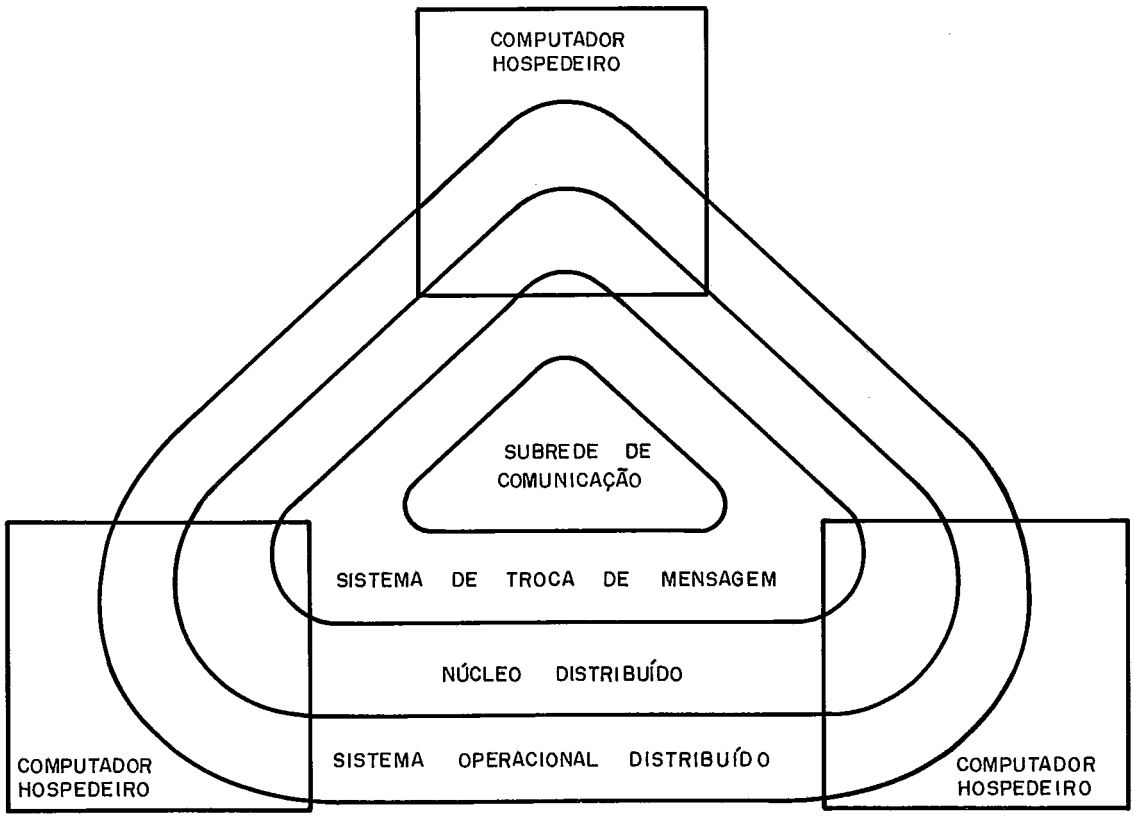


FIGURA V .1 - ESTRUTURA DE UM SISTEMA OPERACIONAL DISTRIBUÍDO

ra tratar os processos em execução, ou mesmo ambicioso, tratando de sincronização entre processos, alocação de recursos, gerenciamento da diretoria do sistema, proteção de objetos, etc.

A questão da proteção tem exercido forte influência na estrutura do sistema operacional e do núcleo, visando a obtenção de proteção efetiva contra ações indesejáveis. Estas ações podem advir do usuário, do sistema operacional, e das partes do próprio núcleo. A proteção tem sido implementada por várias estratégias conjuntas, dentre as quais encontra-se o estabelecimento cuidadoso de direitos de acesso a dados e funções existentes nos vários níveis do sistema.

Pelo fato do núcleo estar posicionado entre o sistema operacional e o hardware, ele deverá ter condições de atender requisições de hardware e de software, além de apresentar mecanismos que permitam uma rápida interação entre elementos do sistema. A chamada de núcleo, por parte dos processos, deverá ocorrer através de chamada de procedimentos do núcleo, ou através da execução de instruções que acabam gerando interrupção. Pelo lado do hardware, os dispositivos chamarão o núcleo através de interrupções. De qualquer maneira, a chamada de núcleo, por hardware ou por software, será sempre encarada como proveniente de uma ocorrência especial considerada como exceção.

Do ponto de vista geral, conforme KUTTI [95], LORIN e DEITEL [108], BACON [11], e SEIDEL e GREBE [151], um núcleo deve apresentar as seguintes funções:

- manipulação das chamadas de núcleo (exceções);
- realização de despachos;
- fornecimento do suporte de E/S;
- gerenciamento de recursos;
- gerenciamento de processos;
- fornecimento do suporte para comunicação e sincronização entre processos;
- outros serviços a serem realizados em tempo de execução.

Antes de se fazer uma rápida descrição de cada uma dessas funções, convém notar que as operações desempenhadas pelo núcleo serão sempre precedidas por uma entrada, provocada por uma chamada de procedimento ou aparecimento de interrupção, e suce

didas por uma saída. Na entrada, o processo que emitir a chamada, ou estiver em execução quando a interrupção aparecer, será suspenso. Dependendo do tipo de chamada de núcleo, o processo suspenso poderá ficar pendente, ser colocado na fila dos processos prontos, ou ser bloqueado em uma fila à espera de uma condição. Na saída, depois do núcleo terminar suas atividades, algum processo previamente selecionado pelo despachante, será colocado em execução.

A manipulação de exceções, ou chamadas de núcleo, corresponde à parcela do núcleo encarregada de identificar a origem da chamada e o tipo de serviço requerido, cuidando para que o serviço seja executado por outra parte do núcleo. A origem pode ser tanto um programa que tenha emitido uma requisição de E/S, por exemplo, quanto um dispositivo que tenha provocado uma interrupção, indicando o cumprimento de alguma operação previamente disparada. Esse tratamento inicial é conhecido como pré-processamento de exceção, ou processamento de exceção em primeiro nível, e é realizado com inibição de interrupções. Normalmente, por questões de segurança e simplicidade, todo trabalho do núcleo é realizado com inibição de interrupções. Porém, há situações onde os requisitos do sistema não permitem que o núcleo trabalhe com a interrupção inibida o tempo todo, como é o caso de sistemas que exigem que o núcleo atenda interrupções prioritárias, mesmo antes de terminar o seu trabalho. Nesse caso, o núcleo deverá utilizar pontos seguros, onde a informação de "status" é suficiente para permitir uma nova interrupção e posterior recuperação do processamento inicial, liberando a interrupção nesses pontos. As partes do núcleo incumbidas de desempenhar os serviços requisitados comportar-se-ão como operações primitivas.

Um despacho consiste em escolher e colocar um processo em execução. A escolha de um processo pelo despachante (escalação de curto prazo) poderá recair sobre o processo pendente, ou um processo da fila dos prontos, dependendo de uma série de fatores como tipo de chamada que o núcleo esteja atendendo, situação do sistema, e política de funcionamento. Para não interferir nas políticas de escalação de níveis superiores do sistema, essa política de seleção de processos deverá ser neutra, e considerará as prioridades, quando existirem. O despachante deverá ser executado como uma das últimas atividades do núcleo, antes de comu

tar o processador para um processo. A atuação do despachante na fase final da execução do núcleo visa fazer com que a seleção de um processo seja a mais realista possível, uma vez que a fila dos prontos estará na sua situação final. Isto é justificável pelo fato do atendimento de requisições de serviço, pelo núcleo, poder alterar a fila dos prontos através da inserção de novos processos.

A seção do núcleo que trata de E/S é conhecida por supervisor de E/S, ou programa de controle de E/S. O controle de um conjunto de dispositivos de E/S, incluindo os "timers", dá-se através de aplicações de requisições de operações de E/S, seguidas dos consecutivos atendimentos de suas interrupções. O programa de controle de E/S atua como um programa de serviço de E/S, deixando outras funções como alocação e liberação de dispositivos para serem tratadas por outras partes do núcleo ou do sistema operacional.

O gerente de recursos é uma parte do núcleo que supervisiona recursos de uso comum, tais como memória, arquivos, dispositivos de E/S, etc. Conforme os processos em execução vão precisando desses recursos, eles os vão requisitando ao gerente de recursos que, de acordo com a disponibilidade, os vai alocando aos processos. Conforme os processos detentores dos recursos não mais precisem deles, ou estejam terminando sua execução, o gerente de recursos vai fazendo o recolhimento e incorporação desses recursos ao seu conjunto de recursos livres. O gerenciamento de recursos pode ser encontrado tanto no núcleo quanto no sistema operacional. O gerenciamento de memória, por exemplo, é uma atividade frequentemente encontrada no núcleo, enquanto que o gerenciamento de arquivos e de dispositivos periféricos muitas vezes fazem parte dos módulos do sistema operacional.

O gerente de processos deve manter as informações de cada processo em estruturas apropriadas denominadas descritores, e controlar as filas de condição, onde deverão permanecer os processos que estejam esperando pela ocorrência de algum evento. Além disso, um gerente deve oferecer facilidades para controlar cada processo através de primitivas capazes de criar, destruir, iniciar, e parar processos. Dependendo da implementação, o gerente de processos pode tornar-se bastante complexo, principalmente quando envolver sistemas dinâmicos.

O suporte para comunicação e sincronização entre processos é implementado através de vários tipos de primitivas de comunicação e sincronização, utilizando troca explícita de mensagem, chamada remota de procedimento, detecção de falhas, recuperação de falhas, ligações estáticas, ligações dinâmicas, etc. Num sistema distribuído, a comunicação entre processos locais ou remotos deverá ser realizada da mesma maneira, cabendo ao núcleo a tarefa de descobrir se vai, ou não, precisar usar o subsistema de comunicação.

Quaisquer outras funções que não se enquadrem nas citadas, e que necessitem ser incorporadas ao núcleo, serão classificadas como outros serviços. A configuração inicial de todo o sistema pode ser encarada como atividade especial ou normal do gerente de processos, que pode criar os processos dinamicamente sob o comando de um processo inicial.

V.1.2 - EVOLUÇÃO DO PROJETO DE NÚCLEO

O projeto de núcleo de sistema operacional e de linguagem para programação concorrente vêm sofrendo uma evolução no sentido de serem produzidos mais rapidamente e de forma mais confiável, procurando manter ou melhorar o seu desempenho. Por outro lado, o crescimento e a sofisticação dos sistemas têm exigido núcleos que atendam a sistemas formados por uma UCP, várias UCPs, ou mesmo vários computadores.

Alguns exemplos dos primeiros núcleos de sistemas operacionais desenvolvidos são os seguintes: núcleo do sistema RC 4000 (BRINCH HANSEN [22]); HYDRA - um núcleo para sistema operacional de multiprocessador (WULF et alii [204]); núcleo do sistema Multics (SCHROEDER [147], SCHROEDER et alii [148]); e núcleo para sistema operacional multiprogramado com facilidades para manipulação dinâmica de recursos (SHAW et alii [153]).

A divisão do sistema operacional em núcleo e módulos de níveis superiores fez com que o tratamento dos detalhes de máquina ficasse ao nível do núcleo, enquanto o tratamento da estrutura lógica do sistema ficasse por conta dos módulos superiores. Isto permitiu a introdução de linguagens de alto nível para a programação da estrutura lógica dos sistemas operacionais, res

paldada pela independência de máquina dos módulos superiores.

A utilização de linguagem de alto nível nos projetos de sistemas operacionais teve uma série de implicações:

- fez com que os sistemas ficassem portáteis, exigindo somente a feitura do núcleo para cada tipo de máquina, o que corresponde a uma pequena parcela do sistema operacional;

- diminuiu o esforço de programação, permitindo a elaboração de sistemas em prazo mais curto e com equipes reduzidas;

- aumentou a legibilidade dos sistemas, melhorando a documentação, manutenção, e adaptação do software, bem como sua confiabilidade;

- provocou uma pequena perda de desempenho, decorrente da sobrecarga do código compilado não otimizado dos módulos superiores do sistema operacional, em relação ao código equivalente escrito em linguagem "assembly";

- foi de encontro às necessidades de projeto de sistemas de microcomputadores que, pelo baixo custo do hardware e diversidade de aplicações, exige que o software de sistema seja de produção rápida, aderente a aplicação, e de baixo custo.

Há inúmeros exemplos de sistemas operacionais, programados com o uso de linguagem de alto nível e apoiados por um núcleo escrito em linguagem "assembly". Esses sistemas são classificados em dois tipos:

- sistemas que usam máquina básica formada pelo núcleo e pelo interpretador de um código intermediário. O sistema operacional programado com uma linguagem de alto nível será compilado, gerando código intermediário que será interpretado durante a execução. A implantação do sistema operacional em máquinas diferentes exigirá somente a reprogramação, em "assembly", da máquina básica;

- sistemas que usam um núcleo e o código objeto do restante do sistema operacional gerado pelo compilador para a máquina específica. Nesse caso, o núcleo serve muito mais para adequar o sistema às várias configurações de máquina do que para dar um suporte que permita a independência de máquina do restante do sistema. A implantação do sistema operacional em máquinas diferentes exige a reprogramação, em "assembly", do núcleo, e a existência de compilador da linguagem de alto nível usada na programação do sistema operacional, em cada uma dessas máquinas.

Como exemplo de sistema de microcomputadores que usam máquina básica pode-se citar o sistema Pascal-UCSD, e como exemplo de sistemas baseados em núcleo e código objeto compilado para a máquina específica tem-se o CP/M. Alguns sistemas de maior porte que utilizaram linguagem de alto nível para a programação do sistema operacional são: sistema MUSS (THEAKER e FRANK [182,183]); sistema SOLO (BRINCH HANSEN [24]); sistema THOTH (CHERITON et alii [36]); sistema TRIPOS (RICHARDS et alii [142]); etc.

Com o aparecimento das linguagens para programação concorrente apropriadas para a elaboração de sistemas em ambientes multiprogramados, várias das funções exercidas pelo núcleo foram incorporadas na linguagem. As primitivas de comunicação e sincronização, por exemplo, constituem-se num caso típico dessa incorporação. Assim, para a execução de qualquer programa escrito com essa linguagem, passou a ser necessário que existisse, além do compilador, um suporte em tempo de execução conhecido como o núcleo da linguagem. Nesta nova estrutura, o compilador passou a analisar também a estrutura de concorrência e suas primitivas, o que antes não acontecia. Aqui, o núcleo que deve ser escrito para cada máquina específica serve de base para a execução de qualquer programa concorrente, inclusive de vários sistemas operacionais programados com essa linguagem.

Dentre as linguagens para programação concorrentes suportadas por núcleo estão: Módula (WIRTH [200]); Pascal Concorrente (BRINCH HANSEN [23]); Menyma (KOCH e MAIBAUM [89]); ADA (ICHBIAH [71], WEGNER [197]); SR (ANDREWS [6,8]); Star Mod

(COOK [37]); etc.

É consenso que o projeto de um núcleo não é tarefa fácil. A responsabilidade do projetista é grande pelo fato do núcleo ser a parte mais crítica do sistema operacional, influenciando fortemente a sua eficiência e confiabilidade. As dificuldades de projeto são decorrentes da alta complexidade do conceito de núcleo, de seu comportamento dependente do tempo, e da insuficiência de ferramentas de programação usadas na sua implementação. A linguagem "assembly", que tem servido para programar muitos núcleos, não suporta qualquer espécie de programação estruturada.

A linguagem Módula-2 (WIRTH [201]) surgiu nesse contexto, permitindo um tratamento homogêneo do sistema operacional em todos os seus níveis, inclusive do núcleo. Esta linguagem, pelo fato de permitir acesso aos registradores da máquina através da declaração dos mesmos como variáveis com endereços fixos, oferece ao programador todos os elementos para cuidar dos detalhes de máquina, usando uma linguagem de alto nível.

Assim, a programação de um núcleo complexo pode ser bastante simplificada de forma a torná-lo fácil de ser escrito, de ser lido, e de ser modificado e verificado. Em comparação com um núcleo escrito em linguagem "assembly", o núcleo escrito em Módula-2 pode apresentar condições favoráveis com relação a tamanho e velocidade, além de explorar as outras vantagens das linguagens de alto nível. Um exemplo de projeto de um núcleo simples escrito em Módula-2 é apresentado em HOPPE [70], onde se procura ilustrar as vantagens citadas. Outro exemplo nesse sentido é apresentado em LEBLANC et alii [100], onde é descrito o projeto e a implementação de um núcleo da linguagem para programação distribuída Star Mod, programado em sua maior parte com um subconjunto dessa mesma linguagem. O principal objetivo desse projeto de núcleo foi obter uma implementação eficiente e portátil. O projeto baseou-se numa estrutura hierárquica, na medida em que primeiramente foi desenvolvido um núcleo para programação concorrente, servindo de suporte para a linguagem Star Mod. Em seguida, essa linguagem foi usada na elaboração do núcleo para programação distribuída.

O aumento do número de linguagens para programação concorrente e o aparecimento de novos elementos e de novas técnicas

para tratar a programação de sistemas distribuídos têm preocupado os projetistas desse tipo de sistema. Os motivos dessa preocupação estão relacionados de um lado com a dificuldade de conseguir que tudo isto esteja a disposição, e de outro com a responsabilidade de se fazer a escolha mais adequada. Para contornar esse problema, já existem algumas iniciativas no sentido de aproveitar-se as boas linguagens de programação já existentes, em vez de se fazer novas propostas de linguagens com características de concorrência. Essas iniciativas consistem na utilização de pacotes de primitivas, correspondendo ao núcleo, juntamente com linguagens já existentes, através de módulos de interface devidamente projetados para tal. Desta forma, o projetista poderá usar os comandos da linguagem e as primitivas do pacote, conseguindo as facilidades necessárias para programar sistemas distribuídos. Para isto, ele deverá fazer uso de um pré-compilador que traduza o seu programa em instruções da linguagem mais simples e chamadas de primitivas do pacote. Nesse sentido, já existem algumas propostas e experiências como a implementação de chamada remota de procedimentos realizada por BIRREL e NELSON [18], e outras, relacionadas em KIRNER et alii [87], FANTECHI et alii [52], e KATAGUIRI [80].

V.1.3 - QUESTÕES DE PROJETO

O desenvolvimento de núcleo apresenta questões de projeto dentro de um amplo espectro, envolvendo desde a definição dos objetivos gerais, até particularidades do ambiente. Para manter um certo nível de generalidade, somente as questões gerais de projeto serão consideradas, embora possam estar relacionadas com o desenvolvimento de projetos específicos.

Dentre as diversas propriedades, que um núcleo de sistemas operacionais deve apresentar, cabe ressaltar três propriedades gerais: universalidade; passividade; e indivisibilidade (KAMIBAYASHI et alii [79]). A universalidade, por considerar que um núcleo é um conjunto de primitivas e funções universais usado na elaboração de sistemas operacionais, deve assegurar que as facilidades contidas no núcleo sejam independentes de aplicações. A passividade está associada ao fato dos módulos do nú

cleo entrarem em funcionamento somente quando forem acionados por alguma chamada. A indivisibilidade é a propriedade que garante que cada módulo do núcleo será executado somente como um todo.

Outras considerações sobre o projeto e objetivos a serem alcançados na implementação foram citadas e discutidas em vários trabalhos, envolvendo aspectos como simplicidade, compactação, eficiência, confiabilidade, proteção, separação de mecanismos e política, transparência de acesso a rede, etc (BUSTARD e HOARE [32], WULF [204], KRONENTAL [93], PRASAD et alii [133], RASHID e ROBERTSON [137]). Aspectos como simplicidade, compactação, e eficiência atuam no sentido do núcleo ser de fácil entendimento, ocupando o menor espaço possível, e gastando um tempo mínimo para sua execução. Admitindo-se que o hardware possua uma estruturação adequada para um funcionamento confiável, é necessário complementá-lo com um software confiável, a começar principalmente pelo núcleo. A confiabilidade do núcleo requer que o programa esteja correto, e que ele seja capaz de detectar e recuperar erros. A proteção é conseguida com esforços conjugados no sentido de estabelecer as restrições necessárias para a obtenção de integridade na execução de operações requisitadas em níveis superiores. As restrições deverão ser aplicadas uniformemente na estruturação do sistema. A separação de mecanismos e política procura colocar cada coisa no seu lugar, além de contribuir com os objetivos anteriores do projeto de núcleo como simplicidade, compactação, eficiência, etc. A política corresponde a forma de tratamento das atividades e elementos do sistema, podendo variar conforme a aplicação, o que justifica a sua colocação fora do núcleo. Em sistemas distribuídos, a localização dos recursos deve ser transparente para facilitar o transporte de processos de uma máquina para outra, e a decomposição de tarefas com a colocação das partes fora dos limites de uma máquina.

Muitos projetos de núcleo têm dado uma ênfase especial em confiabilidade e proteção, condições essenciais para a elaboração de sistemas operacionais seguros (POPEK e KLINE [129]). Os núcleos desenvolvidos com esses objetivos são denominados núcleos seguros. Os esforços realizados no sentido de obter-se esse tipo de núcleo têm procurado minimizar o seu tamanho e complexidade, propiciando as condições para a aplicação de métodos de

verificação de programas (WALKER et alii [194], POPEK e KLINE [129]).

Para que um sistema seja considerado seguro, é necessário que haja uma verificação do sistema implementado, ou demonstração de que o mesmo corresponde a realização correta de sua especificação segura. Para isto, tem-se utilizado técnicas de especificação formal, seguidas da elaboração do programa e conseguente verificação (WALTER et alii [195], POPEK e FARBER [128], SCHROEDER [147], SILVERMAN [162], MILLEN [117]). Embora essa abordagem tenha sido considerada no desenvolvimento do núcleo seguro do sistema Multics (SCHROEDER [147], SCHROEDER et alii [148]), o projeto desse núcleo surgiu da evolução do módulo supervisor desse sistema, em vez de ser concebido naturalmente.

A evolução do supervisor do sistema Multics em um núcleo seguro baseou-se na adoção de uma estratégia geral, que procurou isolar um conjunto mínimo de funções capaz de dar suporte ao sistema, e também encontrar uma forma de estruturar o núcleo, visando facilitar a verificação de sua correção. A estratégia adotada consistiu em produzir um núcleo seguro através de três ações complementares:

- remover do supervisor aquelas funções que não necessariamente precisassem ficar no núcleo, e nem ser protegidas nesse nível;

- reestruturar os mecanismos que permanecessem no núcleo, no sentido de adequá-los à nova situação;

- dividir o núcleo em diferentes partes protegidas, visando tanto a sistematização da tarefa de elaborá-lo, de acordo com as especificações de segurança do sistema, quanto a sua verificação.

Uma outra abordagem para a elaboração de sistemas seguros (RUSHBY [144]) consiste em conceber qualquer sistema como um sistema distribuído, no qual a segurança seja obtida através da separação física dos seus componentes, e de funções de confiança realizadas por alguns desses componentes. As funções de confiança surgem nos sistemas como elementos que exercem atividades complementares às ações do núcleo, e que também são relacionados

com proteção, dificultando a abordagem anterior. Assim, a verificação de correção deverá ser feita tanto para o núcleo quanto para as funções de confiança. O trabalho do núcleo seguro será criar uma máquina virtual isolada em cada nó do sistema distribuído, e manipular as comunicações entre essas máquinas virtuais. No caso de tratar-se de máquina única, o núcleo seguro deverá permitir que a mesma situação seja criada do ponto de vista lógico.

A grande maioria das características desejáveis de um núcleo, tais como simplicidade, pequeno tamanho, bom desempenho, etc., aliada à proximidade do hardware, faz com que o núcleo seja um forte candidato a ser implementado através de microprogramação. Isto, além de melhorar as qualidades citadas, permite a implementação, na prática, da máquina que corresponde a máquina virtual formada pelo núcleo. Um trabalho nesse sentido, apresentado em KAMIBAYASHI et alii [79], conseguiu aumentar o desempenho das operações do núcleo da ordem de quatro vezes pela implementação das mesmas com o uso de microprogramação. Alguns processos do sistema operacional, aos poucos, também tenderão a ser transformados em microprogramas.

V.1.4 - ESTRUTURAS DE NÚCLEO

Os núcleos de sistemas operacionais têm sido desenvolvidos para ambientes que vão desde microcomputadores até sistemas distribuídos. A estrutura do núcleo está sempre relacionada com a estrutura do hardware e do sistema operacional. Assim, existem núcleos simples para sistemas de pequeno porte, núcleos mais complexos para sistemas multiprogramados e para multiprocessadores, e núcleos múltiplos para redes e sistemas distribuídos.

O núcleo para ambiente de multiprogramação consiste de um programa que fornece o suporte necessário à execução de processos concorrentes num único processador. Por isso, a utilização de recursos simples e facilidades do sistema, como "buffers", inibição de interrupção, etc., tem sido suficiente para a implementação de exclusão mútua, comunicação, e outras tarefas desses núcleos. Alguns exemplos de núcleos para ambiente de multiprogramação são analisados em REES e STEPHENS [140], BRINCH HANSEN

[24], HOLT et alii [69], SEIDEL e GREBE [151], SHAW et alii [153], AMES JR. e OESTREICHER [3], BROWN [31], THORELLI [184], e HARIDI et alii [64].

No caso de núcleo para multiprocessador, a existência de processamento fisicamente paralelo impõe dificuldades adicionais que devem ser resolvidas a nível de hardware e de software. Por exemplo, a inibição de interrupção não será mais suficiente para a imposição de exclusão mútua em todo o ambiente; deverão existir instruções do tipo "test and set" (TST), ou equivalente, que, associadas a posições da memória compartilhada e incorporadas a trechos de programa, consigam implementar exclusão mútua. Numa arquitetura homogênea de multiprocessador, o núcleo poderá ser único, ficando localizado na memória compartilhada para ser chamado pelos processadores, conforme a necessidade. A imposição de exclusão mútua na execução do núcleo garantirá a inexistência de conflitos, fazendo com que somente um processador possa utilizá-lo por vez, enquanto outros têm de esperar para executá-lo, mantendo-se normalmente em espera ocupada. Pelo fato da memória compartilhada ter uma tendência em transformar-se num gargalo do sistema, e considerando que algumas atividades do núcleo referem-se a cada processador em particular, como é o caso do tratamento de interrupções, tem-se adotado arquiteturas de multiprocessador, onde os processadores possuem memória local. Isto passa a permitir que uma parte do núcleo seja colocada em cada memória local, junto ao processador correspondente, ficando na memória compartilhada somente aquela parcela que necessariamente será compartilhada (JOSEPH et alii [78]). Para efeito de melhoria de desempenho, pode ser preferível manter uma cópia do código de todo núcleo em cada memória local, deixando na memória compartilhada somente os dados compartilhados. Alguns exemplos de núcleos para multiprocessadores podem ser vistos em JOSEPH et alii [78], HOLT et alii [69], HOLT [68], NATARAJAN e SINHA [120], PRICE [134], e TOKORO et alii [187].

Uma arquitetura genérica de um sistema distribuído pode ser considerada como um conjunto de várias estações interligadas por uma rede, onde cada estação apresente uma capacidade de atendimento de múltiplos processos. A implementação da estação pode dar-se através do uso de multiprogramação, multiprocessamento,ou mesmo de uma estrutura de rede confinada de alta velocidade. A

implementação da estação como um multiprocessador, onde os processadores contam com alguma memória local, foi utilizada em SHOJA et alii [154] e GARETTI et alii [57], e mostra-se aderente à estrutura da linguagem para programação distribuída SR (ANDREWS [6,8]). A implementação da estação como uma rede confinada de alta velocidade pode apresentar desempenho próximo do esquema que usa multiprocessador, porém com as vantagens da distribuição. A memória compartilhada poderá continuar sendo utilizada, em alguns casos, atuando exclusivamente na comunicação de dados entre os elementos como se fosse uma caixa postal. O núcleo de um sistema distribuído implantado numa arquitetura genérica do tipo mencionado será formado pelo conjunto dos núcleos individuais das estações. Da mesma maneira, o núcleo de cada estação será formado pelos núcleos individuais dos elementos da mesma. Tanto a nível da estação quanto do sistema distribuído, cada núcleo individual deverá conter uma parcela igual para todos do mesmo nível, correspondendo às necessidades de comunicação e atividades gerais, e outra parcela diferenciada, caracterizando o elemento ou a estação em questão. Essa parcela diferenciada apresenta uma ligação muito próxima com as peculiaridades do hardware e do software de cada elemento ou estação.

Dos exemplos de núcleo para sistemas distribuídos, encontrado na literatura, alguns usam estações simplificadas como acontece em POWELL e MILLER [131], RASHID e ROBERTSON [137], BARAK e LITMAN [12], KRAMER et alii [91], e VON ISSENDORFF e GRÜNEWALD [192], enquanto que outros utilizam estações melhor elaboradas (SHOJA et alii [154], GRIMSDALE et alii [61]). Além disso, os núcleos podem manter entre si uma relação descentralizada ou centralizada; a maior parte dos exemplos adota a relação descentralizada, mas em alguns casos, como acontece em CHERITON [35], a estrutura de rede estrela acabou por refletir uma organização do núcleo com a relação mestre/escravos, ficando o mestre no nó central. Noutros casos, quando a estação é implementada como multiprocessador, há uma tendência em refletir-se no núcleo as estruturas de sistemas operacionais de multiprocessadores, que serão vistas neste capítulo. A título de exemplo, a estruturação do núcleo da estação multiprocessadora, citada em SHOJA et alii [154], foi implementada usando a organização mestre/escravos.

V.2 - SISTEMAS OPERACIONAIS DISTRIBUÍDOS

A definição tradicional de sistema operacional como *"um programa de controle para alocação de recursos entre tarefas concorrentes"* descreve somente uma parte das responsabilidades dos sistemas operacionais modernos. Um sistema operacional deve apresentar duas funções básicas: a primeira consiste em criar uma máquina abstrata, transformando um conjunto de recursos básicos de hardware, firmware, e software em um conjunto coerente de objetos abstratos como arquivos, diretorias, relógios, bases de dados, etc.; a segunda consiste em multiplexar e alocar os recursos básicos entre várias tarefas, processos, ou usuários.

Um dos grandes problemas com que os projetistas de sistemas operacionais se deparam está relacionado com o gerenciamento da complexidade de operações em muitos níveis de detalhe, abrangendo desde operações de alta velocidade efetuadas pelo hardware, até operações lentas de software. Uma primeira estratégia para abordar esse problema utiliza a técnica de esconder informações (PARNAS [124], HABERMANN et alii [62]). Esta técnica procura confinar os detalhes de tratamento de classes de objetos dentro de módulos que devem apresentar uma interface bastante clara com os outros módulos. Com isto, se o hardware ou alguma parte do software sofrer alteração, não será preciso reprogramar todo o sistema, bastando alterar algum módulo e eventualmente alguma interface. A extensão deste princípio ao sistema operacional como um todo, através da criação de uma hierarquia de níveis de abstração, tem permitido que as partes do sistema operacional sejam elaboradas em níveis, de forma que em cada nível se possa ignorar os detalhes de implementação dos níveis inferiores. No nível mais elevado do sistema operacional encontram-se as interfaces com o usuário. Assim, o sistema operacional pode ser melhor definido como *"uma máquina virtual caracterizada por três ítens: é formada por um conjunto de extensões de software; funciona sobre o hardware do sistema; e atua como um ambiente de programação de alto nível adequado para a implementação de diversas aplicações"* (BROWN et alii [30]).

V.2.1 - HIERARQUIA NO PROJETO DE UM SISTEMA OPERACIONAL

Adotando-se um modelo de um sistema operacional genérico que incorpore idéias de vários sistemas, inclusive facilidades para processamento distribuído, pode-se estabelecer uma estrutura hierárquica, envolvendo vários níveis. Cada nível comporta-se como o gerente de um conjunto de objetos de hardware ou de software, e incorpora a definição de operações que serão realizadas nesses objetos. Essa estruturação do sistema leva em conta duas regras gerais. A primeira corresponde à hierarquia, que estabelece que cada nível deve acrescentar novas operações à máquina e esconder certas operações contidas em níveis inferiores. Num dado nível só poderão ser utilizadas as operações daquele nível e aquelas selecionadas (visíveis) dos níveis inferiores, mas nunca as operações dos níveis superiores. A segunda regra, relativa a esconder informação, procura esconder os detalhes de como um objeto é implementado e onde ele está armazenado para que ele não possa ser alterado, exceto pela aplicação de operações autorizadas para isto.

A estrutura hierárquica da máquina virtual que compõe um sistema operacional genérico pode ser resumida em quinze níveis (BROWN et alii [30]):

nível 1 - circuitos eletrônicos;
nível 2 - conjunto de instruções;
nível 3 - infraestrutura para procedimentos;
nível 4 - atendimento de interrupções;
nível 5 - manipulação de processos do sistema;
nível 6 - suporte para armazenamento secundário;
nível 7 - suporte para memória virtual;
nível 8 - verificação das capacidades;
nível 9 - comunicação entre processos;
nível 10 - sistema de arquivos;
nível 11 - acesso a dispositivos de E/S;
nível 12 - fluxo de dados de E/S;
nível 13 - manipulação de processos do usuário;
nível 14 - manipulação de diretorias;
nível 15 - interface com o usuário.

Os quatro primeiros níveis correspondem rusticamente ao

hardware, embora tenha alguma associação com o sistema operacional, como ocorre com o atendimento de interrupções. Os oito primeiros níveis são aqueles existentes numa máquina simples já conhecidos em sistemas operacionais mais antigos, requerendo poucas modificações para adequação a sistemas operacionais avançados. Os níveis superiores ao oitavo propiciam ao sistema operacional um relacionamento com um ambiente mais amplo do que aquele oferecido pela máquina simples, permitindo o acesso a dispositivos periféricos e outros computadores através de uma rede. Cabe notar que o núcleo do sistema operacional abrange tipicamente níveis que vão do nível 4 até o nível 9, podendo chegar até o nível 11.

O sistema operacional THE (DIJKSTRA [45]) foi um dos primeiros a serem elaborados com estruturação hierárquica.

V.2.2 - ASPECTOS ORGANIZACIONAIS DE SISTEMA OPERACIONAL

Os sistemas operacionais podem ser classificados quanto a sua aplicação e quanto a sua estrutura. Com relação a aplicação, encontram-se os sistemas operacionais para atendimento monousuário ou multiusuário, para aplicações de uso geral ou uso específico, para processamento em lotes, em tempo partilhado, ou em tempo real, etc. Quanto a sua estrutura, os sistemas operacionais são enfocados como um conjunto de processos cooperativos implementados nos seguintes ambientes: simples; multiprogramado; multiprocessado; e distribuído. Um ambiente simples suporta somente sistemas operacionais de pequeno porte e de estrutura e desempenho limitados, como ocorre com alguns sistemas operacionais para microcomputadores. Os outros três ambientes caracterizam-se por suportar sistemas operacionais que exploram algum paralelismo de execução lógico ou físico, permitindo maior desempenho, maior alcance de projeto, e atendimento a aplicações mais complexas.

Um sistema operacional estruturado sobre um ambiente multiprogramado, apesar de explorar o paralelismo lógico da máquina, pode tratar de aplicações monousuário ou multiusuário. Vários desses sistemas são programados com o uso de linguagens de alto nível apropriadas para programação concorrente. Isto faz

com que os sistemas operacionais sejam implementados como um conjunto de processos concorrentes que se comunicam e se sincronizam através de mecanismos como semáforos, monitores, etc. Os sistemas operacionais SOLO (BRINCH HANSEN [24], POWELI [132]) e Pilot (REDELL et alii [139]) são alguns exemplos voltados ao atendimento monousuário, e programados com o uso das linguagens para programação concorrente Pascal Concorrente e Mesa respectivamente. Os sistemas operacionais MUSIC (GRAEF et alii [60]) e Trio (BRINCH HANSEN e FELLOWS [27]) são alguns exemplos voltados ao atendimento multiusuário, e programados com o uso da linguagem Pascal Concorrente, enquanto que o sistema Thoth (CHERITON et alii [36]), também programado com linguagem de alto nível, é orientado para aplicações em tempo real.

Os sistemas operacionais implementados em ambientes constituídos por multiprocessadores e redes, os quais permitem execução fisicamente paralela, apresentam inúmeras variações. No caso de multiprocessadores que caracterizam-se por serem fortemente conectados, e utilizam normalmente alta velocidade de comunicação, os sistemas operacionais são estruturados tanto de forma repartida quanto repetida em cada nó. As redes, por constituirem sistemas frouxamente conectados e com velocidades mais baixas, apresentam forte tendência para a estruturação do sistema operacional de forma repetida, apresentando variações nos recursos e serviços disponíveis em cada nó.

Um sistema operacional distribuído é visualizado como um sistema operacional integrado que unifica o ambiente de forma transparente ao usuário. As principais características de um sistema operacional distribuído (CASAGLIA [34]) são: o controle é distribuído no sentido de que cada nó deve ter seu próprio gerente, cooperando com o conjunto; os processadores são tratados pelo sistema operacional como um conjunto de recursos; a comunicação entre processos ocorre através de troca de mensagem. Sistemas operacionais distribuídos podem ser encontrados em ambientes formados por redes e também por multiprocessadores que utilizem, além da memória compartilhada, memória local associada a cada nó.

V.2.3 - SISTEMAS OPERACIONAIS DISTRIBUÍDOS EM AMBIENTES MULTIPROCESSADORES

Embora as facilidades e serviços oferecidos por um sistema operacional de multiprocessador sejam parecidos com aqueles fornecidos por sistemas operacionais de ambiente multiprogramado, a presença de diversas unidades de processamento no sistema introduz um novo fator no projeto do sistema operacional, referente a sua organização e operação com relação aos múltiplos processadores. Nesse sentido, tem-se utilizado três organizações básicas de sistemas operacionais de multiprocessador, ou seja, sistema mestre/escravos, sistema com supervisor separado em cada processador, e sistema com supervisor flutuante (ENSLOW JR. [49,50], RODGERS [143]).

A organização mestre/escravos é a mais fácil de ser implementada, por permitir que somente um processador particular denominado mestre possa executar o sistema operacional, enquanto que outros processadores denominados escravos só poderão executar programas de usuário. Os escravos serão atendidos um por vez pelo mestre, através do reconhecimento de suas interrupções que serão enfileiradas. Assim, o sistema operacional não precisa ser reentrante, podendo utilizar estruturas e mecanismos mais simples. No entanto, como o processador mestre pode apresentar uma falha irrecuperável, o sistema como um todo pode ser considerado como vulnerável a falhas catastróficas, cuja restauração exige a intervenção do operador. Além disso, o desempenho do sistema poderá sofrer forte degradação, quando o sistema operacional não der conta do atendimento dos escravos que então ficarão ociosos por algum tempo.

Na organização que utiliza supervisor separado, cada processador possui seu próprio sistema operacional e atende às suas próprias necessidades. Cada processador deve controlar seus recursos dedicados, e manter, além de suas tabelas específicas, algumas das tabelas globais do sistema distribuídas entre os vários supervisores individuais. O acesso a essas tabelas globais deve ser cuidadosamente controlado através de técnicas de exclusão mútua, constituindo-se num mínimo de disputa ao nível do sistema operacional. Esta organização é mais confiável que a anterior, pois uma falha em um processador dificilmente provocará

uma falha catastrófica no sistema.

O sistema com supervisor flutuante, denominado também como sistema simétrico, corresponde à organização de implementação mais complexa e poderosa. O supervisor flutua de um processador para outro, cuidando do sistema como um todo, enquanto os outros processadores podem estar usando as rotinas de serviço do supervisor ao mesmo tempo. Para isto, é necessário que sejam utilizados código reentrante e técnicas de exclusão mútua. Normalmente, os processadores de um multiprocessador são idênticos e fazem parte de um conjunto gerenciado pelo sistema operacional, sendo possível o estabelecimento de um bom balanceamento de carga. Os problemas de disputa podem ficar críticos, principalmente quando muitos processadores estiverem precisando do atendimento do supervisor. Uma forma de minimizar esse problema consiste em dividir os dados globais do sistema em entidades separadas e independentes que possam ser acessadas em paralelo. Este tipo de organização é uma das mais confiáveis, pois havendo falha em algum processador, o sistema operacional fará a sua remoção do conjunto de processadores disponíveis e avisará o operador sobre esta ocorrência. O sistema poderá continuar funcionando com uma potência menor de processamento, enquanto o reparo esteja sendo providenciado, apresentando assim uma degradação elegante.

Sistemas operacionais distribuídos implementados em ambientes multiprocessadores utilizam a organização de supervisor separado em cada nó com estruturas parcialmente repetidas e repartidas. Alguns exemplos desses sistemas são Star OS (JONES et alii [76]) e Medusa (OUSTERHOUT et alii [122]), ambos implementados no sistema multiprocessador Cm*.

O sistema operacional Star OS utiliza troca de mensagem e foi projetado para suportar forças-tarefa, que consistem em conjuntos de processos concorrentes cooperativos, visando atingir objetivos comuns. O próprio sistema Star OS é apresentado como um exemplo de força-tarefa, cujos processos espalhados pelos processadores apoiam-se num núcleo repetido em cada processador. O projeto do sistema procurou abordar com maior profundidade os elementos correspondentes às facilidades e mecanismos.

O sistema operacional Medusa foi desenvolvido, procurando enfatizar os aspectos estruturais, e buscando modularidade, robustez, e desempenho. A estrutura de controle do sistema ope

racional encontra-se distribuída, uma vez que os serviços do sistema são implementados de forma disjunta em processos que se comunicam através de mensagens. Pelo fato desses serviços estarem distribuídos entre os processadores disponíveis, e visando evitar competição ou atrasos no acesso à memória, os acessos diretos são restritos somente a serviços locais; se um serviço não estiver disponível localmente, ele deverá ser requisitado através de troca de mensagem, de forma que o processo destino que contém o serviço funcione como servidor. Da mesma maneira que no Star OS, o núcleo é repetido em cada processador.

V.2.4 - SISTEMAS OPERACIONAIS DISTRIBUÍDOS EM AMBIENTES FORMADOS POR REDES DE COMPUTADORES

Os sistemas operacionais implementados em ambientes formados por redes de computadores podem ser classificados como sistemas operacionais de redes ou sistemas operacionais distribuídos.

V.2.4.1 - SISTEMAS OPERACIONAIS DE REDES

Num sistema operacional de rede, cada máquina mantém seu próprio sistema operacional individual complementado com facilidades de comunicação, permitindo a interação entre as várias máquinas e seus sistemas. Esses sistemas operacionais são utilizados geralmente na conexão de computadores heterogêneos e geograficamente dispersos, no sentido de permitir o compartilhamento implícito ou explícito de recursos (DONNELLEY [46], WATSON e FLETCHER [196]).

A rede ARPANET (McKENZIE et alii [115]) constitui-se num primeiro exemplo típico de sistema operacional de rede, controlando um ambiente de grande porte.

Um segundo exemplo de sistema operacional de rede, este relativamente simples e apropriado para estruturas em anel com conexão ponto a ponto, é o sistema Network (BRINCH HANSEN [25]). Esse sistema foi programado com o uso da linguagem Pascal Concorrente e implementado num anel formado por dois computadores. Cada computador apresenta uma versão local do sistema Network que

é composto por um processo de leitura do barramento de entrada, e por um processo de escrita no barramento de saída. Esses processos se comunicam através de um monitor de "buffer" que faz o papel de fila de saída do nó. Esses processos também apresentam comunicação com os processos locais correspondentes às tarefas de aplicação, utilizando os monitores de entrada e de saída, que servem respectivamente para o encaminhamento de mensagens da rede para tarefas específicas do nó e vice-versa. Numa situação genérica, se uma mensagem não for destinada a um nó, ela será transferida do barramento de entrada para o barramento de saída pelos processos do sistema operacional, passando pela fila de saída do nó.

O terceiro exemplo de sistema operacional de rede desenvolvido recentemente é o sistema MIKE ("Multicomputer Integrated KErnel"), descrito por TSAY e LIU [189]. Este sistema foi implementado num protótipo de rede em anel duplo denominado DDLCN ("Distributed Double-Loop Computer Network"). Tal protótipo contém sete nós, cada um composto por uma interface do anel, implementada com microprocessador, e pelo seu correspondente computador hospedeiro. O sistema operacional de rede foi elaborado com o objetivo de alcançar autonomia cooperativa entre os sistemas locais de computação, funcionando de modo transparente para os usuários da rede. Para isto, cada computador da rede, além de conservar seu sistema operacional original, mantendo-se familiar aos usuários de cada nó, passou a relacionar-se com a rede, como se ela fosse um único sistema de computação integrado. O sistema operacional MIKE consiste de um conjunto de núcleos repetidos, cada um residindo em uma interface do anel, apresentando assim uma grande independência do hardware e do software dos computadores hospedeiros. Esse sistema apresenta uma estruturação hierárquica, consistindo de três níveis correspondentes a: comunicação entre processos; suporte do sistema; e máquina virtual.

O quarto exemplo, consistindo do sistema operacional ZCZOS (ZHONGXIU et alii [205]), ressalta que sua implementação foi baseada no sistema operacional RT 11, adequado aos microcomputadores LSI-11 que formam os nós da rede. Assim, apesar do sistema ZCZOS ter sido denominado sistema operacional distribuído, ele se configura como um sistema operacional de rede. O sistema foi implementado num protótipo, contendo cinco microcomputa

dores LSI-11 interligados. Para a sua programação foi usada a linguagem Módula-2, exceto em uma pequena parte. Além disso, o sistema possui um algorítmo especial de determinação de rota para apresentar independência de topologia de rede. O sistema fornece basicamente funções de comunicação entre computadores, compartilhamento de recursos, alocados ao longo da rede, e facilidades para suportar programação distribuída.

V.2.4.2 - SISTEMAS OPERACIONAIS DISTRIBUÍDOS

Um sistema operacional distribuído compreende um único sistema operacional homogêneo implementado para o ambiente como um todo. Duas abordagens têm sido utilizadas no desenvolvimento desses sistemas: a primeira refere-se a sistemas servidores, e a segunda, a sistemas simétricos ou integrados.

Na abordagem do servidor (BARAK e LITMAN [12]), também denominada abordagem da rede de computadores pessoais (FINKEL [54]), algumas máquinas específicas são designadas para atender certos tipos de serviço do sistema, enquanto outras são utilizadas para a execução de tarefas. Gerenciamento de periféricos, correio eletrônico, etc., são alguns tipos de serviço desses sistemas, cujas aplicações predominam em ambientes de escritório.

Na abordagem simétrica (BARAK e LITMAN [12], também conhecida por abordagem do conjunto de processadores (FINKEL [54]), cada processador da rede apresenta-se como uma estação independente com hardware e recursos de software completos. O sistema operacional distribuído, nesse caso, tem a função de integrar as várias máquinas da rede em uma única máquina virtual que forneça acesso transparente e compartilhamento de recursos. Aplicações muito dispendiosas em termos de potência de computação, como a solução de problemas numéricos de grande porte, têm sido tratadas com a abordagem do conjunto de processadores. As vantagens do uso de sistemas distribuídos simétricos estão relacionadas com melhoria nos seguintes ítens: desempenho; disponibilidade; confiabilidade; e crescimento incremental.

A seguir, alguns exemplos de sistemas operacionais mencionados na literatura serão analisados resumidamente.

— ARACHNE

O sistema operacional distribuído Arachne (FINKEL e SOLOMON [55]), preliminarmente denominado Roscoe (SOLOMON e FINKEL [168]), é um sistema operacional orientado para aplicações de uso geral. Ele foi implementado numa rede de computadores formada por cinco microcomputadores LSI-11, de forma a apresentar ao usuário o comportamento de um único computador de uso geral. Sua programação utilizou na sua maior parte a linguagem C, exceto em uma parcela do núcleo que foi escrita em "assembly".

As características essenciais do sistema são:

- todos os processadores são idênticos, e executam a mesma cópia do núcleo do sistema operacional:
- nenhuma memória é compartilhada, e a comunicação se dá somente por troca de mensagem;
- não se faz nenhuma suposição sobre a topologia de interconexão, a não ser que consiste de uma rede;
- a rede configura-se ao usuário como uma única máquina.

O sistema foi estruturado de maneira que as funções tradicionais fossem fornecidas, não pelo núcleo, mas sim por processos denominados processos de serviço. Cada máquina apresenta os processos de serviço necessários ao seu bom funcionamento, configurando-se assim a utilização do modelo simétrico otimizado.

— SODS/OS

O sistema operacional distribuído SODS/OS (SINCOSKIE e FARBER [163]) tem como objetivo principal permitir a execução de programas descentralizados e com controle distribuído. Para isto, todos os processos executados sob a supervisão do SODS/OS são independentes de localização e comunicam-se através de troca de mensagem, possibilitando a migração transparente de processos. O sistema foi implementado num protótipo formado por dois computadores IBM séries/1 interligados, estando prevista a sua expansão para ser implementado em uma rede local em anel. O sistema é composto de tarefas ou módulos cooperativos, consistindo

de: núcleo; gerente de processos; gerente de filas de mensagens; gerente da memória do usuário; gerente de chamadas do sistema operacional; e gerente da rede. Aproximadamente 95% do código do sistema operacional foi programado com o uso da linguagem C, e o restante com o uso de "assembly". Em princípio, cada estação deve apresentar sua cópia do sistema operacional com pequenas diferenças, devido às peculiaridades locais como a presença de algum periférico, configurando-se a utilização da abordagem simétrica.

– MICROS

O sistema operacional distribuído MICROS (WITTIE e VAN TILBORG [203], VAN TILBORG e WITTIE [190]) é orientado para suportar múltiplos usuários, executando programas concorrentes. O sistema, embora seja adequado para um ambiente de milhares de nós, foi implementado num protótipo, contendo dezesseis microcomputadores LSI-11 interligados frouxamente através de uma rede estensível e reconfigurável. A ligação de cada microcomputador com a rede formada por dois barramentos compartilhados de alta velocidade é feita através de uma interface, que nada mais é do que um processador de comunicação. Cada nó da rede é controlado por uma cópia local do núcleo do sistema, que consiste de um conjunto de processos concorrentes programados em Pascal Concorrente.

O sistema MICROS apresenta uma estrutura hierárquica do ponto de vista lógico, contendo três níveis:

- o primeiro nível conhecido como gerência contém informações que permitam fazer alocação global;
- o segundo nível conhecido como subgerência tem capacidade para alocações regionais;
- o terceiro nível é aquele onde ocorre a execução de tarefas do usuário.

O sistema utiliza gerentes redundantes no topo da hierarquia e ligações latentes entre gerentes do mesmo nível com o objetivo de ajudar na recuperação de eventuais falhas. Cada nó da estrutura hierárquica troca informações de controle e de esta

do somente com os níveis vizinhos da hierarquia. As tarefas de gerenciamento e os programas de usuário são escritos em Pascal Sequencial e são colocados dinamicamente nos nós. A comunicação de um processo com outros, internos e externos ao nó, é feita através de troca de mensagem. A evolução do projeto desse sistema consistiu de sua reprogramação com o uso da linguagem Módula-2.

— AMOEBA

O sistema operacional AMOEBA (TANEMBAUM e MULLENDER [179]) foi desenvolvido para controlar um conjunto de processadores interligados por uma rede. O sistema baseia-se no conceito de serviço implementado em processos servidores, cuja requisição deve ocorrer por meio de mensagens. A estrutura utilizada considera seis níveis de protocolo, onde os dois primeiros, correspondendo ao nível físico e ao nível de enlace de dados, são implementados por meio de uma combinação de hardware e software. O terceiro nível denominado monitor juntamente com os dois primeiros formam o núcleo do sistema operacional. Os outros dois níveis, correspondendo ao nível de transporte e ao nível de chamada do sistema, constituem o sistema operacional, ficando o último nível por conta do usuário.

— CONIC

O projeto CONIC (SLOMAN et alii [164,165,166], KRAMER et alii [91], MAGEE [109]) é responsável por uma abordagem unificada e integrada de desenvolvimento, implementação, e gerenciamento de grandes sistemas distribuídos de controle.

O desenvolvimento do projeto abrange vários ítens:

- uma rede de computadores;
- uma metodologia de desenvolvimento de software de aplicação;
- um sistema operacional distribuído e um sistema de comunicação;
- um sistema de gerenciamento de alterações.

Para esse desenvolvimento utilizou-se a linguagem Pascal, estendida com o conceito de módulo e com primitivas para troca de mensagem. O sistema foi implantado num protótipo de rede, composto por cinco microcomputadores LSI-11 interligados por barramentos assíncronos, formando um anel. Cada estação apresenta uma cópia do núcleo que fornece o suporte a múltiplas tarefas, cuidando da troca de mensagem que implementa a comunicação entre as tarefas. O sistema operacional de cada estação mantém um conjunto de serviços relativos ao tratamento de módulos e de seus elementos associados, e coopera com as outras estações, formando um sistema operacional distribuído. Esta estrutura corresponde àquela obtida com a abordagem de sistema operacional distribuído simétrico. O núcleo foi implementado na sua maior parte com o uso da linguagem Pascal, ficando uma pequena parte programada em "Assembly". O sistema operacional foi desenvolvido com o uso da linguagem Pascal estendida.

- DEMOS/DP

O sistema operacional distribuído DEMOS/DP (POWELL e MILLER [131] é uma versão do sistema operacional DEMOS (BASKETT et alii [14]) estendido para funcionar num ambiente distribuído. Esse sistema foi implementado numa rede de microcomputadores Z8000, cuja comunicação é feita por troca de mensagem. Cada processador possui uma cópia do núcleo, contendo os objetos primitivos do sistema. Embora cada núcleo mantenha seus próprios recursos (UCP, memória, canais de E/S, etc.), todos os núcleos cooperam no fornecimento de um mecanismo de troca de mensagens confiável e transparente com respeito à localização dos processos. Independentemente de pequenas diferenças, os processadores do sistema podem ser considerados iguais, podendo suportar os mesmos tipos de serviços, o que facilita a migração de processos. Muitas das funções do sistema são implementadas em processos servidores, sendo acessadas através do mecanismo de comunicação. Esta estrutura configura a adoção da abordagem do servidor na organização do sistema operacional.

- LOCUS

O sistema operacional distribuído LOCUS (WALKER et alii

[193], MUELLER et alii [119]) é um sistema operacional simétrico, compatível com o UNIX, e voltado para aplicações de uso geral. O sistema foi implementado num protótipo formado por um conjunto de dezessete computadores VAX/750 interligados através de um barramento tipo Ethernet.

Durante a elaboração do projeto, procurou-se atingir os seguintes objetivos:

- fazer com que o desenvolvimento das aplicações distribuídas não fossem mais difíceis do que programar máquinas simples;

- procurar obter operações altamente confiáveis e disponíveis através da exploração de redundância e distribuição.

O sistema oferece alto grau de transparência da rede, ao mesmo tempo em que apresenta bom desempenho e facilidades como repetição automática e flexível de armazenamento a nível de arquivos. Cada computador possui sua cópia do núcleo e do sistema operacional. A programação do sistema foi realizada com a linguagem C.

- PERSEUS

O sistema operacional PERSEUS (ZWAENEPOEL e LANTZ [206]) foi desenvolvido para ser introduzido em vários tipos de máquinas interligadas por redes tradicionais, caracterizando-se como um sistema operacional distribuído. Para isto, sua estrutura foi baseada na existência de um núcleo e sua implementação foi feita com o uso da linguagem Pascal, ficando assim altamente portátil. Somente uma parte do núcleo foi programada em "assembly". A maioria das funções do sistema operacional encontra-se em processos servidores, cuja execução ocorre no modo usuário. As facilidades de comunicação entre processos têm como base as trocas de mensagem. O núcleo que deve ser repetido em cada nó foi desenvolvido com o objetivo de ser mínimo, podendo assim ser executado com interrupção inibida. Pelo fato do sistema distribuído estar previsto para ser implantado numa rede heterogênea, e não necessitar de todas as funções em todos os nós, foi adotado o modelo do servidor que utiliza a alocação de servidores diferen

tes em nós diferentes. Isto permitiu que duplicações desnecessárias pudessem ser evitadas, implicando na redução da quantidade de código em cada nó.

- MOS

O sistema operacional distribuído MOS ("Multicomputer Distributed Operating System"), discutido em BARAK e LITMAN [12], é um sistema operacional simétrico voltado para aplicações de uso geral em tempo compartilhado. O sistema foi desenvolvido desde o início para um ambiente de múltiplos computadores, constituído por um conjunto homogêneo de máquinas autônomas. O protótipo contou com quatro computadores PDP-11 interligados por uma rede local.

A utilização de um ambiente homogêneo permitiu que o sistema apresentasse as seguintes características:

- facilidade para migração de processos;
- transparência da rede;
- estrutura intercambiável;

Os dois objetivos principais considerados no desenvolvimento do projeto envolveram a redução da complexidade, e a obtenção de bom desempenho. O sistema caracteriza-se pela apresentação dos seguintes ítens: multiplicidade de recursos; transparência da rede; controle descentralizado; autonomia; disponibilidade; e cooperação. A elaboração do sistema operacional foi feita de acordo com o modelo do objeto, configurando-se como um gerente de objetos abstratos. Os únicos objetos ativos do sistema são os processos que podem alterar o estado dos outros objetos. Cada objeto reside exatamente em uma máquina, podendo ser manipulado diretamente somente pelo núcleo nela localizado. Cada máquina possui sua cópia idêntica do núcleo, o qual é dividido em três partes: o núcleo inferior; o elemento de ligação; e o núcleo superior. O núcleo inferior implementa os objetos residentes em cada máquina, sendo o único a possuir total conhecimento dos objetos locais e a responsabilidade pela sua integridade. O elemento de ligação atua como intermediário entre o núcleo superior e inferior, sendo o único a distinguir operações locais e operações remotas, devendo acessar os circuitos de comunicação

em caso de necessidade. Além disso, o elemento de ligação é quem mantém o conhecimento atualizado da configuração da rede. O núcleo superior tem a função de conduzir as chamadas do sistema e manter o ambiente dos processos, enquanto faz com que a rede fique escondida do programa do usuário.

– SODA

O sistema operacional distribuído SODA ("Simplified Operating System for Distributed Applications"), discutido em KEPECS e SOLOMON [83], é um sistema operacional constituído basicamente por um núcleo distribuído, contendo várias facilidades necessárias a um sistema distribuído. O sistema foi elaborado para funcionar num ambiente formado por um conjunto de nós interligados por um barramento. Cada nó contém um processador de cliente (hospedeiro) e um processador de núcleo (interface com a rede) ligados a uma memória compartilhada. O projeto foi desenvolvido, tendo como objetivo fornecer facilidades simples e potentes. O núcleo desse sistema pode receber chamadas do cliente, reconhecendo dez comandos diferentes que são executados atomicamente. Os argumentos dos comandos são passados através da memória compartilhada. Esses comandos referem-se funcionalmente aos seguintes assuntos: comunicação; sincronização; nomeação; e controle de processo. A implementação do sistema SODA foi realizada por simulação.

C A P Í T U L O VI

COMUNICAÇÃO E SINCRONIZAÇÃO EM SISTEMAS DISTRIBUÍDOS

Os aspectos de comunicação e sincronização de um sistema distribuído encontram-se embutidos nos vários níveis do protocolo OSI-ISO (DAY e ZIMMERMANN [43] e TANENBAUM [178]), ou equivalente, sendo mais relacionados com a estrutura da rede nos níveis mais baixos, e abstraindo-se dessa estrutura nos níveis mais altos. O enfoque desse trabalho terá seu alcance limitado aos níveis de rede e de transporte, na medida em que considera a transferência de pacotes e de mensagens respectivamente.

Nesse contexto, a topologia e a forma de funcionamento da rede são de importância fundamental, uma vez que acabam exercendo forte influência nas características de comunicação e sincronização pela utilização de algorítmos de rota, detecção e recuperação automática de falhas, etc., determinando assim a visão operacional da rede.

Uma rede pode ser considerada de acordo com os seguintes pontos de vista (SOI e AGGARWAL [167]):

- ponto de vista funcional;
- ponto de vista do projetista (ou de comutação);
- ponto de vista do gerente (ou da topologia);
- ponto de vista operacional (ou do roteamento);
- ponto de vista da comunicação entre os componentes do sistema;
- ponto de vista combinado da rota com a topologia.

Embora o escopo deste capítulo esteja mais ligado com os três últimos pontos de vista, todos eles serão abordados, a seguir, com a ênfase adequada.

VI.1 - FORMAS DE VISUALIZAÇÃO DE UMA REDE E SEUS ASPECTOS DE COMUNICAÇÃO

VI.1.1 - VISÃO FUNCIONAL DA REDE

Do ponto de vista funcional, uma rede pode apresentar-se de três maneiras:

- como suporte para interações entre um usuário e um computador hospedeiro remoto;
- como suporte para interações entre computadores hospedeiros independentes;
- como parte de um sistema integrado correspondente a um sistema distribuído.

VI.1.2 - VISÃO DO PROJETISTA DA REDE

Do ponto de vista do projetista, uma rede tende a ser classificada de acordo com a forma de comutação ou técnica de interconexão dos computadores. Desta maneira, tem-se redes de comutação de circuito, de comutação de pacotes, de comutação de mensagens e de comutação híbrida (circuito-mensagem).

VI.1.3 - VISÃO DO GERENTE DA REDE

Do ponto de vista do gerente da rede, os aspectos topológicos são os que mais influenciam na sua classificação. Nesse sentido, as redes apresentam-se como centralizadas, descentralizadas, ou distribuídas.

VI.1.4 - VISÃO OPERACIONAL DA REDE

Desse ponto de vista, as redes apresentam-se de acordo com o método de roteamento dos elementos utilizados na comunicação, tais como: pacotes; mensagens; etc.

A estratégia de roteamento baseia-se num conjunto de regras que determinam os trajetos, sobre os quais mensagens ou pacotes devem fluir de um lugar para outro.

O roteamento pode ser classificado de várias maneiras, dependendo dos fatores que estiverem sendo considerados. Quanto ao estado das informações sobre a rede, tem-se roteamento fixo, ou estático, e roteamento adaptativo, ou dinâmico (DAVIES et alii [42] e MARTIN [114]). Quanto à localização e à forma de recolhimento das informações sobre a rede, o roteamento pode ser considerado como centralizado, descentralizado, ou distribuído (SOI e AGGARWAL [167] e MARTIN [114]). Quanto à tomada de decisão, tem-se roteamentos que usam algorítmos determinísticos, algorítmos estocásticos, e algorítmos de controle de fluxo (SOI e AGGARWAL [167] e DAVIES et alii [42]).

O roteamento fixo, ou estático, é caracterizado por utilizar informações fixas a respeito da rede, podendo escolher um trajeto dentre várias possibilidades. Em geral, a rota é determinada pelo critério do menor caminho, sendo insensível às variações do nível de tráfego de informações através da rede.

O roteamento adaptativo, ou dinâmico, caracteriza-se por utilizar informações variáveis a respeito da rede, baseadas na intensidade de tráfego de informações ao longo do tempo.

O aparecimento de falhas permanentes, ou de longa duração, poderá fazer com que as informações acerca da rede utilizadas no roteamento fixo, sejam alteradas para adequar-se à nova situação. No caso de roteamento adaptativo, o aparecimento de falhas deverá ser considerado como mais um elemento de variação a ser levado em conta. O eventual desaparecimento da falha também deverá ser absorvido nos dois tipos de roteamento. O roteamento adaptativo apresenta maior facilidade para o tratamento de falhas e para a atividade dinâmica de inserção e de retirada de elementos da rede.

O roteamento centralizado baseia-se na existência de um nó, denominado centro de controle da rota, para o qual todos os nós enviam periodicamente informações sobre tráfego e topologia. Essas informações servem para a geração de tabelas atualizadas de roteamento que são encaminhadas aos vários nós da rede.

O roteamento descentralizado caracteriza-se por ser realizado em cada nó com informações ali disponíveis, podendo apre

sentar algum grau de adaptação a mudanças de tráfego e de topologia.

O roteamento distribuído baseia-se na utilização de informações disponíveis em cada nó, obtidas e atualizadas periodicamente através da interação com os outros nós.

Os roteamentos que usam algorítmos determinísticos obtêm a rota de acordo com uma regra específica previamente definida, levando em conta as condições ideais para se alcançar o destino.

Os roteamentos que usam algorítmos estocásticos determinam a rota de acordo com regras de decisão probabilística, levando em consideração a topologia e as estimativas de tempo para se alcançar o destino. Nesse caso, as informações que servem de base para a determinação das rotas são atualizadas periodicamente.

Os roteamentos que usam algorítmos de controle de fluxo referem-se à capacidade da rede em recusar a aceitação de tráfegos adicionais, quando já estiver sob pressão de um tráfego intenso, e for incapaz de realizar desvios de tráfego para rotas alternativas, no sentido de evitar congestionamento.

As técnicas de roteamento podem ser classificadas de acordo com os três tipos de algorítmo mencionados, conforme a figura (VI.1).

O detalhamento dessas três técnicas consta em seguida.

- Técnicas de Roteamento que Utilizam Algorítmos Determinísticos:

No roteamento por inundação ("flooding"), cada nó recebe uma mensagem e a retransmite nas linhas de saída até que a mesma retorne ao ponto de origem como uma confirmação. O roteamento fixo baseia-se numa topologia fixa e numa intensidade de tráfego conhecida. O roteamento com tráfego repartido permite que o tráfego de informações flua por mais de um trajeto com probabilidades diferentes de utilização. O roteamento do observador ideal emprega o algorítmo do menor tempo de atraso, procurando minimizar o tempo de transferência entre a origem e o destino. O roteamento do menor trajeto procura estabelecer uma rota que envolva o menor número de ligações, podendo não corresponder ao menor tempo de transferência.

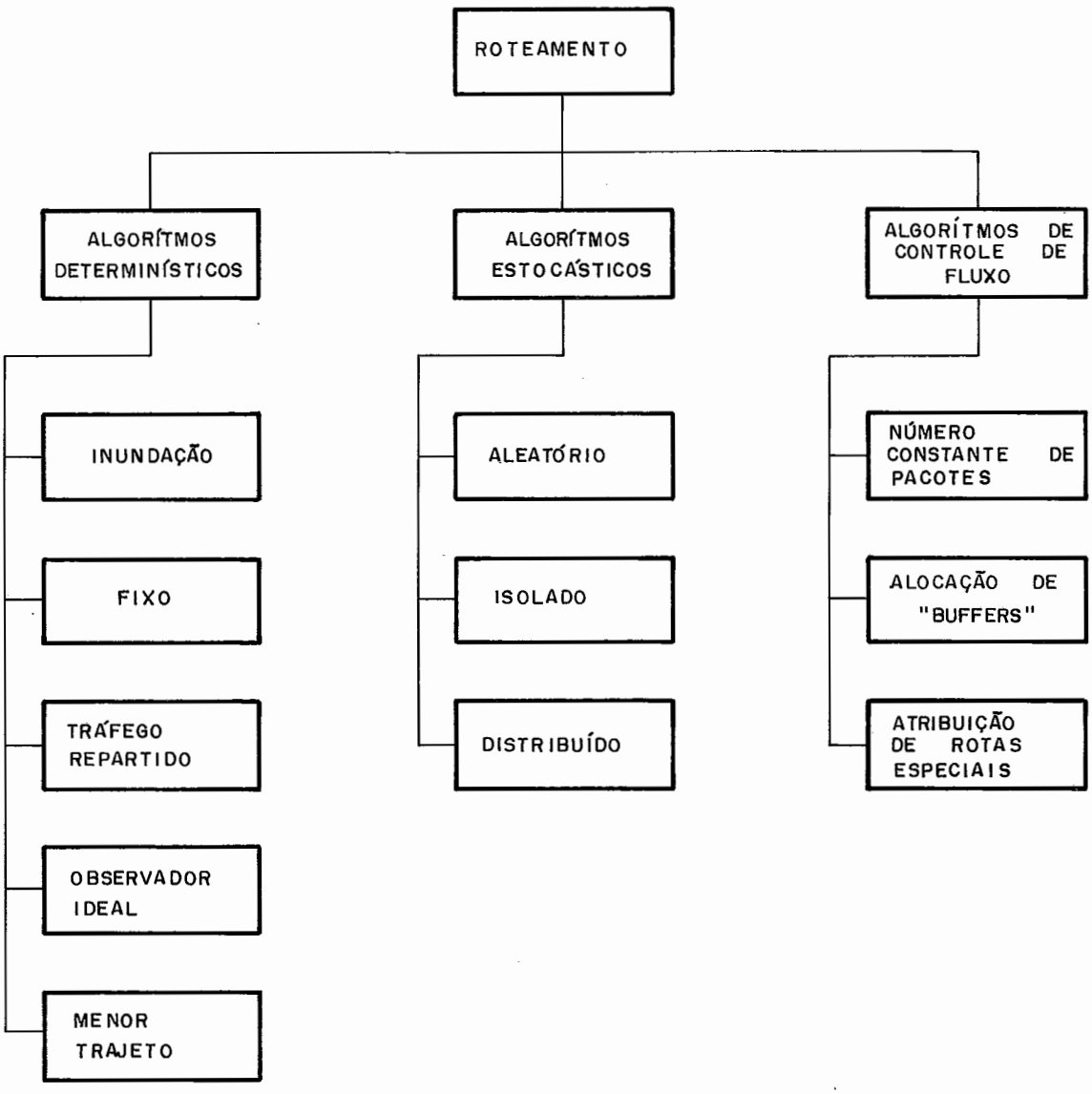


FIGURA VI.1 - CLASSIFICAÇÃO DAS TÉCNICAS DE ROTEAMENTO COM RELAÇÃO A TOMADA DE DECISÃO

– Técnicas de Roteamento que Utilizam Algorítmos Estocásticos:

No roteamento aleatório, cada nó procura mandar as mensagens recebidas para a frente, escolhendo aleatoriamente qualquer ligação, podendo no entanto submeter-se a pequenas influências de direcionamento. O roteamento isolado, por sua vez, caracteriza-se pelo fato do nó tomar decisão por conta própria ba

seada nos parâmetros de experiências anteriores, ou na estratégia de livrar-se o mais rápido possível da mensagem recebida; para isto, a mensagem deve ser colocada na fila mais curta que possa levar ao destino. O roteamento distribuído é dependente da troca de informação de atrasos observados entre os nós, podendo usar a abordagem do vetor de atraso mínimo, ou a abordagem da delimitação da rede em áreas, visando reduzir a quantidade de informação trocada dentro da mesma.

- Técnicas de Roteamento que Utilizam Algorítmos de Controle de Fluxo:

O roteamento numa rede com número constante de pacotes apresenta uma forma de controle de fluxo baseada no número de pacotes vazios, controlando efetivamente a aceitação do tráfego da rede. O roteamento com alocação prévia de "buffers" no destino faz com que o fluxo de mensagens seja condicionado à existência de "buffers" livres, reservados por requisições que antecedem o encaminhamento de mensagens, eliminando assim a sua perda eventual. Para situações, onde a alocação prévia de "buffers" não é suficiente, pode-se lançar mão do roteamento com atribuição de rotas especiais baseadas tanto nas informações de "status" dos nós adjacentes, quanto na intensidade de tráfego encontrada pelo nó nos momentos anteriores.

VI.1.5 - VISÃO DA COMUNICAÇÃO ENTRE OS COMPONENTES

Do ponto de vista das características e dos componentes de comunicação, as redes podem atuar de três maneiras: servindo como suporte para o compartilhamento de recursos; permitindo a implementação de computação distribuída; ou sendo usadas para apoiar comunicações remotas. No caso de servir como suporte para o compartilhamento de recursos, a rede deve oferecer as facilidades de comunicação necessárias, de forma que os recursos remotos sejam disponíveis aos nós como se fossem locais. No caso de computação distribuída, os programas e processos cooperativos em execução em computadores diferentes da rede comunicam-se e trocam informações, tendo em vista tarefas abrangentes. No caso de apoiar comunicações remotas, a rede permite a interligação

eficiente dos usuários a sistemas remotos.

VI.1.6 - VISÃO HÍBRIDA DO ROTEAMENTO E TOPOLOGIA

Outra maneira de classificar as técnicas de roteamento consiste em situá-las de acordo com a topologia da rede, conforme consta na figura (VI.2).

As técnicas de roteamento usadas em redes centralizadas oferecem um desempenho satisfatório, embora mantenham restrições naturais como maior propensão a um colapso total no caso de falha do nó central, além de apresentarem pouca flexibilidade para ajustamentos a variações de carga. Numa situação de tráfego estável, os algorítmos de roteamento centralizado apresentam melhor desempenho do que aqueles com roteamento distribuído.

Por outro lado, as técnicas de roteamento distribuídas apresentam uma imunidade ao colapso total, e funcionam de forma adaptativa, refletindo em cada instante ou período de tempo a situação da rede.

VI.2 - MECANISMOS DE COMUNICAÇÃO E SINCRONIZAÇÃO

A comunicação entre os processos de um sistema distribuído, independentemente de localização, deve dar-se através de troca de mensagens. Para isto, o sistema deve apresentar mecanismos de comunicação e sincronização capazes de realizar a sincronização entre os processos e a troca de informações entre eles.

Existem vários tipos de mecanismos e muitas variações na forma de executar troca de mensagens, dependendo de como são realizados o sincronismo, o endereçamento, a utilização de "buffers", etc.

O mecanismo de comunicação e sincronização entre processos pode servir-se de primitivas simples como *envia* e *recebe*, ou de estruturas mais elaboradas como, por exemplo, *chamada remota de procedimento*. Quanto à sincronização, as primitivas podem ser *bloqueantes* (síncronas) e *não bloqueantes* (assíncronas). O

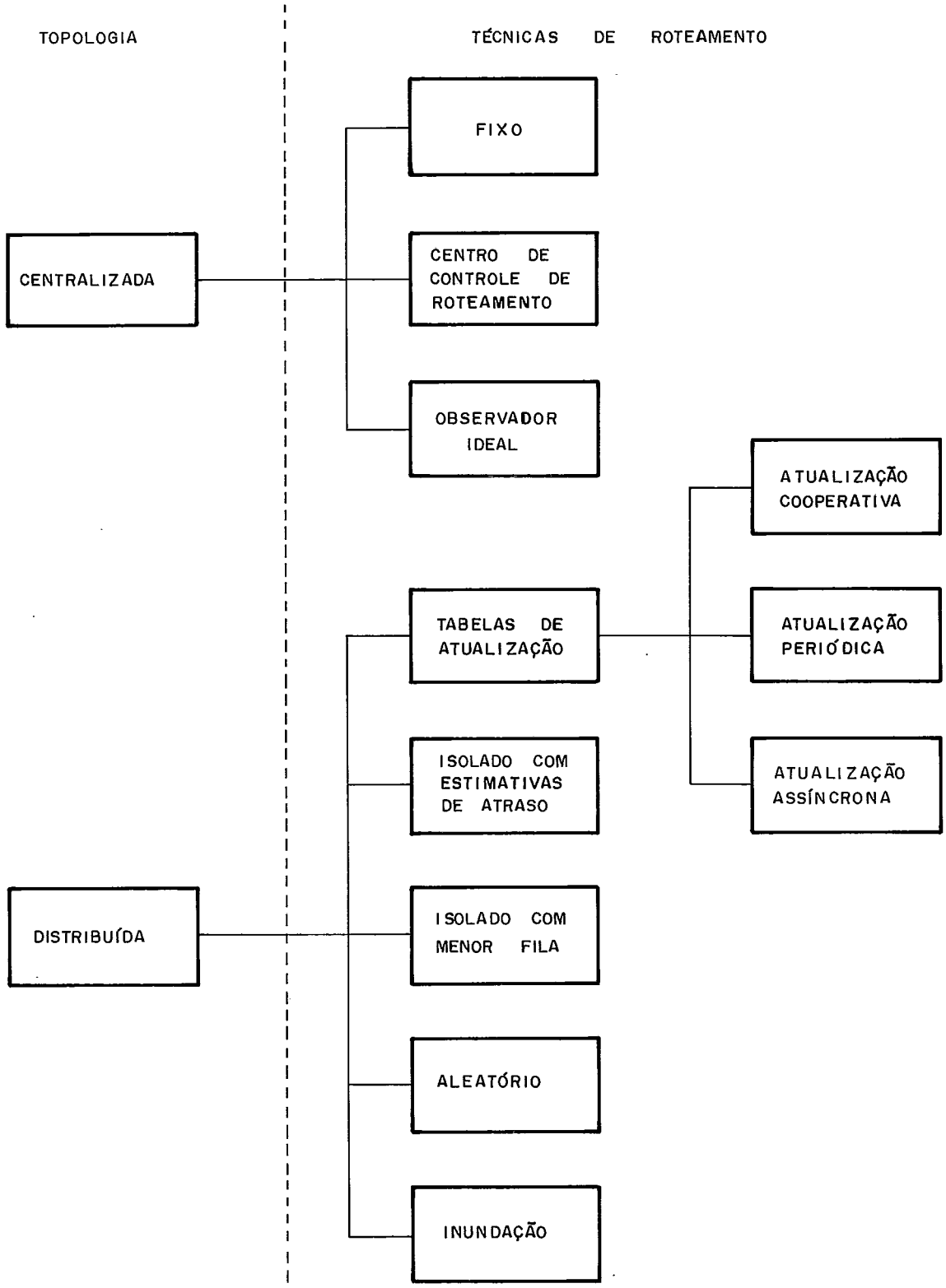


FIGURA VI. 2 - CLASSIFICAÇÃO DAS TÉCNICAS DE ROTEAMENTO COM RELAÇÃO A TOPOLOGIA DA REDE

endereçamento pode ser *explícito* ou *implícito*, *simétrico* ou *assimétrico*. A comunicação entre processos pode ocorrer através de esquemas *um para um*, *multicanal*, ou *difusão*. Com relação à utilização de "buffers", estes poderão encontrar-se *na origem*, *no destino*, ou em *ambos os lugares*. Os *portos*, pela flexibilidade e aplicação em criação e destruição dinâmicas de processos, terão um tratamento à parte. Finalmente, os mecanismos podem *não envolver manipulação de falhas*, podem *detectar falhas*, e podem *recuperar falhas* automaticamente.

VI.2.1 - TIPOS DE IMPLEMENTAÇÃO DOS MECANISMOS DE COMUNICAÇÃO E SINCRONIZAÇÃO

VI.2.1.1 - MECANISMOS SIMPLES

Os mecanismos simples são aqueles que utilizam primitivas básicas como *envia* e *recebe*, dispostas convenientemente nos processos de maneira a permitirem sincronização e comunicação.

A primitiva *envia* tem a função de transferir uma informação do processo origem (emissor) para o processo destino (receptor), enquanto a primitiva *recebe*, por sua vez, atua no sentido de coletar a referida informação. O comportamento dos processos emissor e receptor, ao executarem as respectivas primitivas, dependerá do tipo de sincronização e do endereçamento adotado.

O "rendez-vous" utilizado na linguagem CSP (HOARE [67]) é um exemplo de mecanismo simples obtido pela colocação, em processos diferentes, de um par de primitivas inter-relacionadas de emissão e de recepção, de forma a realizarem a sincronização e a troca de mensagens.

VI.2.1.2 - MECANISMOS MAIS ELABORADOS

Os mecanismos mais elaborados são aqueles associados a determinadas estruturas, como procedimentos ou trechos de programa, cuja implementação equivale a conjuntos de primitivas básicas, adequadamente colocadas nos processos para a obtenção de

sincronização e comunicação. É o caso do "rendez-vous" estendido utilizado na linguagem ADA (ICHBIAH et alii [71], WEGNER [197]), e a chamada remota de procedimento utilizada na linguagem DP (BRINCH HANSEN [26]), SR (ANDREWS [6,8]) e outras. No ítem seguinte, esses mecanismos serão analisados e discutidos.

VI.2.2 - SINCRONIZAÇÃO

A sincronização refere-se à restrição no andamento dos processos. Se uma primitiva é bloqueante (síncrona), significa que o processo que a estiver executando ficará bloqueado até que a operação correspondente seja executada por completo. Se uma primitiva é não bloqueante (assíncrona), o processo que a estiver executando continuará em andamento tão logo ela seja concluída localmente.

VI.2.2.1 - PRIMITIVAS BLOQUEANTES (SÍNCRONAS) E NÃO BLOQUEANTES (ASSÍNCRONAS)

As primitivas básicas *envia* e *recebe* podem apresentar-se de forma bloqueante e não bloqueante.

A primitiva *envia bloqueante*, ao ser executada pelo processo emissor, fará com que este fique bloqueado até que o processo receptor que pode ser remoto receba a mensagem enviada; só então é que o processo emissor poderá ter continuidade.

No caso da primitiva *envia não bloqueante*, bastará que a mensagem a ser enviada seja encaminhada ao núcleo para que o processo emissor tenha continuidade; paralelamente à execução do processo emissor, o núcleo fará com que a mensagem seja encaminhada ao processo receptor que pode ser remoto, independentemente dele estar sincronizado ou não. Se o processo receptor não estiver pronto para receber a mensagem, mas o sistema possuir "buffers" disponíveis para o armazenamento de mensagens, estas serão retidas até serem recebidas. No caso de não haver "buffers", ou se os "buffers" existentes estiverem todos ocupados, a mensagem será perdida.

A primitiva *recebe bloqueante*, ao ser executada pelo processo receptor, fará com que este fique bloqueado até o rece

bimento da mensagem, após o qual poderá ter continuidade.

A primitiva *recebe não bloqueante* procurará receber alguma mensagem que porventura esteja pronta para ser recebida, fazendo com que o processo receptor tenha continuidade, tanto no caso de sucesso na recepção quanto no caso de insucesso.

A implementação dessas primitivas, além de exigir algum protocolo que permita a sincronização e a transferência das mensagens, será dependente da estruturação e alocação de "buffers" e de elementos auxiliares no sistema.

De acordo com MANNING et alii [110], existem quatro maneiras de combinar-se essas primitivas, ou seja:

a) Envia Bloqueante - Recebe Bloqueante (Eb-Rb);
b) Envia Bloqueante - Recebe Não Bloqueante (Eb-Rnb);
c) Envia Não Bloqueante - Recebe Bloqueante (Enb-Rb);
d) Envia Não Bloqueante-Recebe Não Bloqueante (Enb-Rnb).

O caso (Eb-Rb) é chamado de totalmente síncrono, enquanto que o caso (Enb-Rnb) é totalmente assíncrono. Os outros dois casos são denominados semi-síncronos.

Essas quatro combinações de primitivas de emissão e recepção constituem mecanismos simples de comunicação e sincronização.

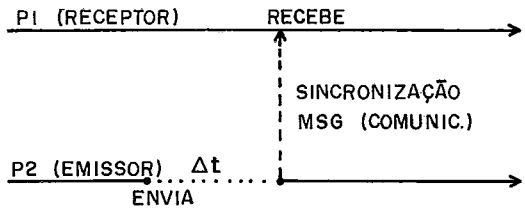
VI.2.2.2 - MECANISMOS DE COMUNICAÇÃO SÍNCRONOS

Os mecanismos de comunicação síncronos conseguem realizar a sincronização e transferência de dados de maneira simples, sem que haja a obrigatoriedade de uso de "buffers", uma vez que a transmissão poderá ser feita somente quando ocorrer a sincronização entre os processos.

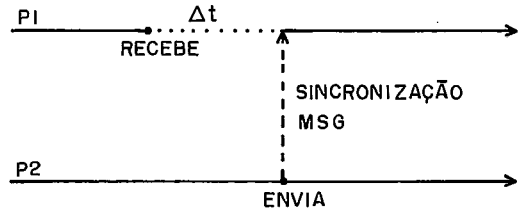
Conforme SEGRE e KIRNER [150], há três tipos de mecanismos de comunicação síncronos:

- "rendez-vous";
- "rendez-vous" estendido;
- chamada remota de procedimento.

A figura (VI.3) mostra de forma simplificada o comportamento dos processos no tempo, quando se utiliza esses três meca

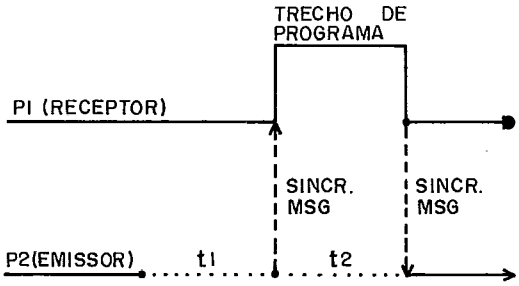


a1) EMISSOR ANTECIPADO

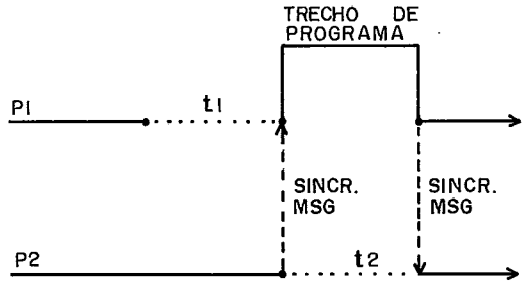


a2) RECEPTOR ANTECIPADO

a) "RENDEZ - VOUS"

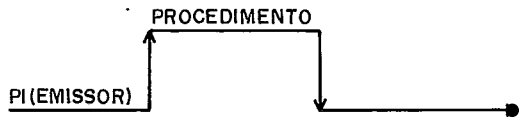


b1) EMISSOR ANTECIPADO

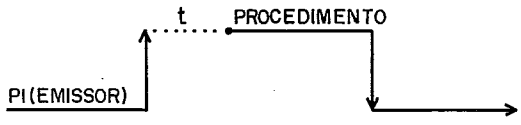


b2) RECEPTOR ANTECIPADO

b) "RENDEZ - VOUS" ESTENDIDO

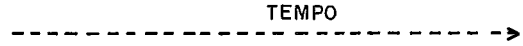


c1) PROCEDIMENTO DISPONÍVEL



c2) PROCEDIMENTO OCUPADO

c) CHAMADA REMOTA DE PROCEDIMENTO



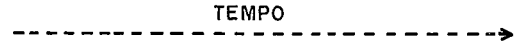


FIGURA VI . 3 - COMPORTAMENTO DOS PROCESSOS AO UTILIZAREM OS MECANISMOS DE COMUNICAÇÃO SÍNCRONOS

nismos de comunicação.

O "rendez-vous", como já foi mencionado, é obtido através de primitivas *envia* e *recebe* bloqueantes colocadas em processos distintos; a execução dessas primitivas em tempos diferentes fará com que o processo que executar a primitiva antes do outro fique bloqueado até que ocorra a sincronização entre os dois processos, e a consecutiva transferência de mensagem; depois disso, ambos os processos continuarão seu andamento em paralelo. No caso extremo de ambas primitivas serem executadas ao mesmo tempo, os dois processos gastarão o tempo necessário para a sincronização e a transferência da mensagem, para, em seguida, continuarem simultaneamente sua execução normal.

O "rendez-vous" estendido caracteriza-se por apresentar uma estrutura de comunicação, onde um processo consegue comandar a execução de um trecho de programa previamente estabelecido pertencente a outro processo, envolvendo sincronização e eventualmente troca de mensagem. O processo requisitante, ao realizar a chamada de um ponto de entrada de outro processo, ficará bloqueado, enquanto não receber uma resposta de término desse trecho. Para isso, a chamada deve ser sincronizada com o ponto de entrada do trecho de programa do outro processo. Assim, qualquer dos dois processos que atingir o seu ponto de sincronização em primeiro lugar ficará à espera de que o outro atinja o seu ponto para que a sincronização possa acontecer. Depois de sincronizado, o processo receptor executará o trecho de programa, findo o qual ocorrerá nova sincronização, indicando o término do trecho de programa, e permitindo que ambos os processos possam dar andamento simultâneo às suas execuções.

A chamada remota de procedimento, por sua vez, apresenta uma estrutura de comunicação na qual um processo pode comandar a execução de um procedimento situado em outro processador. O processo chamador deverá ficar bloqueado até que o procedimento chamado termine. Tanto a chamada quanto o retorno podem envolver troca de mensagem, conduzindo parâmetros. Se o procedimento chamado for reentrante, ou não estiver em operação, ele será acionado tão logo chegue o comando de chamada; se o procedimento não for reentrante e estiver em operação, então a sua execução poderá ser disparada logo que a operação em andamento terminar. A chamada remota de procedimento pode ser encarada como

um comando de execução de procedimento que, ao ser devidamente interceptado pelo sistema tanto na chamada quanto no retorno, fará com que o procedimento seja executado, recebendo e enviando parâmetros associados com outro processo situado noutro computador (CARPENTER e CAILLIAU [33]).

Tanto o "rendez-vous" estendido quanto a chamada remota de procedimento estão relacionados com a transação do tipo "request-reply" apresentada por KRAMER et alii [92]. Esse tipo de transação consiste na transferência de duas mensagens: uma de requisição emitida pelo processo requisitante para o processo de serviço; e outra de resposta emitida pelo processo de serviço para o processo requisitante. Com a utilização de primitivas adequadas, esse tipo de transação poderá ser transformado em "rendez-vous" estendido ou em chamada remota de procedimento desde que, neste último caso, a requisição seja endereçada a um procedimento.

O esquema de "rendez-vous", além de ser o mais simples e flexível, mantém os processos comunicantes independentes um do outro a menos da sincronização. Não existe transferência de controle de um processo para outro, o que permite a inter-relação de um grupo de processos de forma cooperativa. Poderá ocorrer que um processo envie uma mensagem para um segundo processo que, depois de processá-la, a enviará para um terceiro; este último poderá terminar o processamento, remetendo o resultado de volta para o primeiro que irá esperar ou estará esperando pela resposta em algum ponto.

Os esquemas de "rendez-vous" estendido e chamada remota de procedimento são mais complexos e restritivos do que o "rendez-vous", apresentando no entanto mais segurança. Nesses casos, quando uma chamada for executada, haverá a transferência de controle de um processo para outro. Isto poderá gerar o estabelecimento de uma hierarquia na organização de processos, onde o processo chamador estará em nível mais alto do que o chamado.

VI.2.2.3 - MECANISMOS DE COMUNICAÇÃO ASSÍNCRONOS

A maior parte dos mecanismos de comunicação assíncronos correspondem às combinações das primitivas *envia* e *recebe* nos ca

sos semi-síncronos e totalmente assíncronos. O assincronismo será assegurado pela presença de pelo menos uma primitiva *envia* ou *recebe* não bloqueante.

A utilização de primitivas não bloqueantes permitem altas taxas de paralelismo entre os processos pelo fato deles não exigirem sincronização. Para que o processo que estiver executando a primitiva possa continuar sua execução bastará que, no caso do uso da primitiva *envia não bloqueante*, a mensagem tenha sido entregue ao sistema; no caso da utilização da primitiva *recebe não bloqueante* bastará que tenha havido uma tentativa de recepção com ou sem sucesso.

De acordo com LISKOV [103], as primitivas não bloqueantes são consideradas mais flexíveis e de nível mais baixo, uma vez que podem ser usadas para implementar as primitivas bloqueantes.

A primitiva *envia não bloqueante* é muito útil em situações, onde um processo necessita enviar mensagens para muitos destinatários sem preocupar-se se eles irão ou não recebê-las; é o caso de uma difusão de informação de alarme. A primitiva *recebe não bloqueante*, por outro lado, é bastante útil em situações, onde se queira receber mensagens imprevisíveis como acontece com os alarmes.

Os mecanismos de comunicação assíncronos podem ser utilizados conforme o tipo de aplicação. A combinação *envia bloqueante - recebe não bloqueante* pode ser utilizada como parte de uma estrutura do tipo cliente/servidor, onde cada cliente procura enviar uma requisição de serviço ao servidor, enquanto o servidor procura atender a todos os clientes nominalmente num sistema de varredura. A combinação *envia não bloqueante - recebe bloqueante* pode ser usada numa estrutura de processos cooperativos, de forma que um primeiro processo envia uma mensagem de interesse exclusivo do segundo e continua sua execução sem necessidade de interação naquele ponto. A combinação totalmente assíncrona, ou seja, *envia não bloqueante - recebe não bloqueante* pode ser usada num sistema, onde vários processos cooperam aleatoriamente com um único. Neste caso, os emissores não precisarão ficar bloqueados, havendo somente a necessidade do receptor fazer uma varredura de recepção sem bloquear-se.

Além das primitivas assíncronas consideradas,

GENTLEMAN [58] apresenta outros três tipos de primitivas: *envia condicional*; *recebe condicional*; e *responde* ("reply"). A primitiva *envia condicional* emitirá sua mensagem somente se o receptor já estiver pronto para recebê-la, caso contrário haverá o retorno de uma indicação de falha. Da mesma maneira, a primitiva *recebe condicional* só receberá uma mensagem se ela já estiver disponível, caso contrário haverá o retorno de uma indicação de falha. A primitiva *responde* equivale a uma primitiva *envia não bloqueante* executada no receptor, visando mandar a resposta ao emissor.

VI.2.3 - ENDEREÇAMENTO

As primitivas *envia* e *recebe* são definidas de forma resumida como:

envia (destino, mensagem, outros parâmetros);
recebe (origem, mensagem, outros parâmetros).

Um aspecto importante nessa estrutura de comunicação refere-se à técnica usada para a identificação do emissor/receptor da mensagem. Na linguagem CSP (HOARE [67]), isto é feito explicitamente em ambos os processos através de referência mútua. Uma situação mais comumente encontrada consiste em ter-se um endereçamento unilateral, onde o emissor indica o receptor, mas o receptor dispõe-se a comunicar-se com qualquer emissor. Esta abordagem de recepção de qualquer emissor é adequada para o relacionamento clientes/servidor, particularmente quando se utiliza chamada remota de procedimento; ao servidor caberá apenas a execução do procedimento requisitado, sendo desnecessário o conhecimento antecipado do cliente; o identificador do cliente deverá ficar retido implicitamente para ser usado por ocasião do retorno da chamada remota de procedimento.

VI.2.3.1 - ENDEREÇAMENTO IMPLÍCITO E EXPLÍCITO

O endereçamento implícito, conforme GENTLEMAN [58], corresponde ao uso de ligações previamente estabelecidas entre os

processos do sistema. Estas ligações, quando estabelecidas na fase de criação dos processos, e mantidas durante toda a vida do sistema, contribuem para uma situação estática muito limitada, raramente aceitável na prática.

O endereçamento explícito baseia-se na utilização de nomes, endereços, e outros atributos dos processos, explicitamente referenciados nas primitivas de comunicação. O endereçamento explícito, quando usa nomes e endereços, é classificado como direto, e quando usa portos ou elementos equivalentes é denominado indireto (ANDREWS e SCHNEIDER [9]).

A linguagem CSP (HOARE [67]) constitui-se num exemplo de utilização de endereçamento explícito direto, onde as primitivas *envia* e *recebe* devem possuir a indicação recíproca dos processsos envolvidos.

Um exemplo de endereçamento explícito indireto pode ser encontrado na linguagem CLU (LISKOV [103], WILLIAMSON e HOROWITZ [199]) que se utiliza de portos como base para a comunicação entre processos.

A simplicidade do endereçamento explícito direto torna-o apropriado para situações de configuração estática, enquanto que a flexibilidade do endereçamento explícito indireto faz com que o mesmo seja utilizado em configurações dinâmicas.

VI.2.3.2 - ENDEREÇAMENTO SIMÉTRICO E ASSIMÉTRICO

O endereçamento simétrico, conforme MARTELLI e TARINI [113] e AHAMAD e BERNSTEIN [2], caracteriza-se por exigir que as primitivas utilizadas numa comunicação contenham a indicação recíproca da origem e do destino da mensagem, podendo envolver o nome dos processos ou seus portos.

O endereçamento assimétrico, por outro lado, exige que exista somente a identificação do destino da mensagem, sendo feita pela primitiva existente no processo emissor; o processo receptor deverá receber mensagens sem o conhecimento prévio de suas origens. A primitiva *recebe de qualquer um* ("receive any"), que implementa este tipo de endereçamento, foi discutida por GENTLEMAN [58] e CUNHA et alii [39], e incorporada na linguagem MENYMA (KOCH e MAIBAUM [89]).

O endereçamento simétrico, por apresentar uma rigidez muito grande na ordem de comunicação entre os processos, e por exigir que o receptor conheça as origens das mensagens que irá receber, é contraindicado em aplicações do tipo clientes/servidor, principalmente em situações onde os clientes são variáveis.

O endereçamento assimétrico, embora não satisfaça a todas aplicações possíveis, é adequado à aplicações do tipo clientes/servidor, pois permite não determinismo na ordem de recepção de mensagens, e faz com que o servidor possa ser utilizado como rotina de biblioteca mesmo em configurações dinâmicas.

A linguagem CSP utiliza comunicação simétrica, enquanto DP e ADA constituem-se em exemplos de linguagem que incorporam comunicação assimétrica.

VI.2.3.3 - COMUNICAÇÃO ENTRE GRUPOS DE PROCESSOS

Em sistemas distribuídos, o conceito de agrupamento de processos atuando conjuntamente é de vital importância. Esses processos costumam ser utilizados para: obtenção de processamento paralelo; aumento de disponibilidade de dados; redução de tempo de resposta; compartilhamento de recursos; aumento de confiabilidade; etc. Nesse sentido, os processos de um grupo interagem entre si e com processos de outros grupos através de várias formas de comunicação que, segundo BACON [11], são as seguintes:

- comunicação um para um;
- comunicação vários para um;
- comunicação um para vários;
- comunicação vários para vários.

Outra maneira de abordar as várias formas de comunicação entre grupos de processos consiste em visualizá-las, levando em conta as possibilidades de emissão. Esta abordagem, cobrindo parcialmente a classificação citada, é apresentada por FRANK et alii [56], referindo-se a:

- emissão única ("unicast");
- difusão ("broadcast");
- emissão múltipla ("multicast").

A comunicação um para um, ou de origem e destino únicos, ocorre quando o nome ou a localização das duas partes envolvidas numa conversação deve ser de conhecimento mútuo. A maneira apropriada de implementar esse tipo de comunicação consiste em usar o endereçamento simétrico como acontece na linguagem CSP. Este tipo de comunicação corresponde à emissão única ("unicast").

A comunicação vários para um, ou de origem múltipla, baseia-se numa situação onde vários processos origem podem comunicar-se com um processo destino. É o caso da comunicação de múltiplos clientes com um único servidor. O uso de primitivas assimétricas faz-se necessário, uma vez que o servidor deverá funcionar sem o conhecimento prévio do cliente que irá atender, embora cada cliente conheça o servidor para o qual enviará mensagem. Se o endereçamento explícito indireto for utilizado, o servidor poderá ser substituído sem que os clientes percebam. Esse tipo de comunicação é encontrado nas linguagens DP e ADA.

Na comunicação um para vários, ou de destino múltiplo, um processo origem pode comunicar-se com vários processos destino. Este tipo de comunicação pode ser usado para a notificação da ocorrência de alarmes e de condições excepcionais para várias partes do sistema, e corresponde à emissão múltipla ("multicast"). A implementação desse tipo de comunicação poderá dar-se através do uso de várias técnicas que serão analisadas no próximo ítem. A comunicação um para todos, conhecida por difusão ("broadcast"), constitui-se num caso particular de emissão múltipla.

Na comunicação vários para vários, ou de origem e destino múltiplos, vários processos origem podem comunicar-se com vários processos destino simultaneamente. Este tipo de comunicação pode ser usado, por exemplo, para resolver o problema da comunicação de múltiplos clientes com vários servidores repetidos. Uma solução desse problema pode ser obtida, usando-se portos compartilhados nos processos servidores e comunicação vários para um.

VI.2.3.4 - TÉCNICAS DE EMISSÃO MÚLTIPLA ("MULTICAST") E DIFUSÃO ("BROADCAST")

Em algumas aplicações, torna-se necessário que um processo envie pacotes ou mensagens para vários processos de um sistema ("multicast"), ou para todos ("broadcast"). De acordo com FRANK et alii [56] e TANENBAUM [178], este tipo de comunicação pode ser implementado, usando as seguintes técnicas:

- inundação ("flooding");
- endereçamento separado;
- endereçamento multidestino;
- endereçamento repartido;
- árvore de encaminhamento.

A técnica da inundação caracteriza-se por propagar pacotes idênticos em todos os canais, visando atingir todos os processos de um determinado grupo. A maior desvantagem desta técnica está na inundação da rede com pacotes repetidos. Para ser prática, esta técnica deve funcionar com pacotes que possuam uma vida limitada no sistema, evitando assim multiplicações indefinidas do pacote. Além disso, os pacotes devem estar associados a um número de sequência para evitar-se repetições na utilização dos mesmos.

A técnica de endereçamento separado consiste em encaminhar um pacote separado para cada destino. Para isto, cada membro deverá manter uma cópia da lista de todos os membros do grupo. A principal desvantagem dessa técnica situa-se no alto volume de transmissão e nos atrasos, pois várias cópias, diferindo apenas no campo de endereço destino, deverão fluir pelo mesmo canal. Como o atraso está ligado com a emissão sequencial de várias cópias do pacote, esta técnica será adequada somente para grupos com poucos destinatários.

A técnica de endereçamento multidestino funcionará como o endereçamento separado, exceto quando um subconjunto dos pacotes precisar seguir por uma mesma rota; aí, um único pacote encabeçará os vários destinos relativos a esse subconjunto, e seguirá em frente até que alcance os destinos ou tenha que bifurcar, quando então será desmembrado para continuar por rotas diferentes. Para poder conduzir vários destinos, o cabeçalho do pacote

deverá conter um campo de endereço destino que pode ser implementado com uma lista de endereços ou mapa de "bits". O tratamento de um cabeçalho variável e a decisão de repetição de pacotes para fazê-los chegar aos diversos destinos fazem com que esta técnica imponha uma sobrecarga de processamento considerável.

A técnica de endereçamento repartido corresponde à combinação do endereçamento separado com o endereçamento multidestino. Assim, os computadores destino são repartidos em subconjuntos, de forma que os pacotes são enviados primeiramente para cada subconjunto, sendo depois distribuídos para os computadores daquele subconjunto. Esta técnica é muito útil para difusão em interconexão de redes.

A técnica da árvore de encaminhamento baseia-se no uso de uma árvore de profundidade, ou de distância, do nó emissor aos outros nós do grupo. Cada nó da rede, conhecendo quais de suas linhas pertencem à árvore, poderá transferir o pacote da linha de chegada para as outras linhas que fazem parte do grupo, contribuindo para que os destinos sejam alcançados. A principal desvantagem desta técnica está na necessidade dos nós conhecerem a árvore, quando muitos algorítmos de rota não possuem tal conhecimento. No caso dos nós não terem conhecimento da árvore de encaminhamento, pode-se usar uma técnica que se aproxima do comportamento da técnica em discussão. A técnica alternativa consiste em propagar os pacotes nas outras linhas, sempre que estiverem passando por ali pela primeira vez, ou em descarregá-los, se já tiverem passado por ali.

As técnicas mencionadas podem ser utilizadas tanto em redes baseadas em canais ponto a ponto quanto em canais de difusão, sendo adequadas a um ou a outro tipo em cada caso. A aplicação dessas técnicas em sistemas com canais de difusão é mais acentuada nos casos de sistemas com múltiplos canais de difusão interligados.

VI.2.4 - UTILIZAÇÃO DE "BUFFERS"

Na comunicação entre dois processos de um sistema distribuído que estiverem localizados em computadores diferentes, a mensagem pode passar pelo seguinte trajeto: ser preparada num

"buffer" do processo origem; ser remetida ao "buffer" do núcleo associado a este processo; passar por vários "buffers" da rede; chegar ao "buffer" do núcleo associado ao processo destino; e finalmente, ser colocada no "buffer" do processo destino. Alguns desses "buffers" poderão não existir, dependendo do tipo da rede e da comunicação utilizados.

Os "buffers" têm dupla função: servem para controlar situações de congestionamento na rede, e também para controlar o fluxo de informação.

O controle de congestionamento faz-se necessário, quando existe a possibilidade de degradação do desempenho da subrede de comunicação, devido à presença de muitos pacotes de informação e à dificuldade, ou impossibilidade, de armazenar todos eles.

O controle de fluxo, por sua vez, deverá existir quando a emissão de mensagens, ou de pacotes, precisar ser controlada para evitar que a chegada dessas informações seja mais rápida do que a capacidade de sua manipulação por parte do receptor.

Assim, enquanto o controle de fluxo atua nos elementos terminais da comunicação (processos), o controle de congestionamento deve ocorrer na parte intermediária da comunicação (subrede). Muitas vezes, o controle de fluxo é utilizado para eliminar, ou para minimizar problemas de congestionamento.

As estratégias de tratamento de congestionamento, de acordo com TANENBAUM [178] são as seguintes:

- pré-alocação de "buffers" ao longo do circuito virtual por onde circularão os pacotes de mensagem, de forma que, numa rede de conexão ponto a ponto, por exemplo, a transferência do pacote de um "buffer" para o "buffer" posterior libere o avanço do pacote do "buffer" anterior;

- descarregamento de pacotes, quando não houver espaço disponível;

- restrição do número de pacotes permissível na rede;

- uso de técnicas de controle de fluxo de informação;

- suspensão temporária de entrada de pacotes.

O congestionamento também poderá advir do aparecimento de impasses ("deadlock") entre os elementos da rede, provenientes do uso indiscriminado dos "buffers". O impasse pode ser direto, envolvendo somente dois elementos, ou indireto, envolvendo vários elementos formando um ciclo. A solução desse problema pode ser obtida com o uso de algorítmos de roteamento e alocação de "buffers" livres de impasse. Os impasses também podem aparecer ao nível da comunicação entre processos, e serão tratados no próximo ítem.

Para evitar que um receptor receba mensagens mais rapidamente do que possa consumir, deve-se utilizar técnicas de controle de fluxo. Uma primeira abordagem consistiria em dotar o processador, que contém o processo receptor, com "buffers" em quantidade suficiente para absorver todas as mensagens que por ventura sejam dirigidas simultaneamente ao processo receptor. Pelo fato do número de "buffers" no destino poder crescer a níveis insuportáveis, devido a limitações de memória, esta solução apresenta-se como inviável.

Outra alternativa seria manter as mensagens em "buffers" situados nos seus locais de origem sob controle dos processadores respectivos, e enviar aos processos receptores somente os resumos dessas mensagens que serão denominados anúncios. Numa situação de pior caso, quando todos os "buffers" dos locais de origem estiverem preenchidos e estiverem sendo encaminhados para um receptor específico, pode-se definir o número máximo de anúncios que um processador deverá possuir. Como o anúncio é de tamanho bastante reduzido, em comparação com a mensagem formada por muitos pacotes, os "buffers" de anúncio, mesmo em número elevado, não deverão ocupar muito espaço de memória. Esta solução foi utilizada de forma simplificada por VON ISSENDORFF e GRÜNEWALD [192], e ampliada por KIRNER e SANTOS [86].

A terceira abordagem para o controle de fluxo seria encaminhar as mensagens, correndo o risco de serem aceitas ou não; se o receptor não estiver em condições de recebê-las, elas serão descarregadas, devendo ser retransmitidas um certo número de vezes até serem recebidas ou até vencer um limite de tempo.

Por não envolverem perda de mensagem, as duas primeiras abordagens funcionam bem, inclusive com cargas elevadas, enquanto que a terceira funciona muito bem com cargas baixas, mas so

fre um alto índice de degradação em situação de cargas elevadas, devido a perdas contantes de mensagem. As três abordagens apresentam desvantagens: a primeira usa muita memória para "buffer" nos receptores; a segunda, por necessitar de um protocolo bem elaborado, apresenta uma sobrecarga de tempo; e a terceira é muito vulnerável pelo fato de depender do nível de carga da rede e do comportamento estatístico do sistema.

Neste trabalho escolheu-se a segunda opção porque ela apresenta viabilidade de implementação e comportamento confiável e bem determinado, embora com menor desempenho. A discussão e implementação dessa opção serão feitas no próximo capítulo.

VI.2.5 - FALHAS NA COMUNICAÇÃO

Um sistema distribuído deve tolerar falhas físicas e lógicas de forma que, ocorrendo algumas delas, o sistema continue operando mesmo em situação de degradação. Nesse sentido, o sistema de comunicação deve possuir um esquema de detecção de erros capaz de permitir que somente as mensagens corretas sejam encaminhadas aos seus destinos. Isto demonstra um comportamento de comunicação binário, fazendo com que a mensagem ou seja entregue corretamente ou nunca seja entregue.

De acordo com KRAMER et alii [92], uma transação do tipo "rendez-vous" estendido ou chamada remota de procedimento pode falhar pelas seguintes razões:

- perda da mensagem de chamada pelo sistema de comunicação;
- não aceitação da chamada no ponto de recepção, devido falha no processador ou no próprio procedimento;
- falha no procedimento ou no trecho de programa equivalente durante sua execução;
- perda da mensagem de retorno pelo sistema de comunicação.

Qualquer uma dessas falhas fará com que o processo chamador fique bloqueado indefinidamente à espera da mensagem de

resposta. Para poder contornar esse problema, o sistema deve ter capacidade de detectar falhas na transação, e de abandoná-la ou abortá-la. Além disso, visando preservar as situações de pior caso no tempo de resposta, o processo chamador deverá estabelecer um limite de tempo ("time-out") para a realização da transação, a partir do qual também ocorrerá o abandono ou aborto da mesma.

Do lado do processo chamado, no caso de "rendez-vous" estendido, o processo não deverá ficar suspenso indefinidamente esperando por uma chamada. Isto poderá ser resolvido com a utilização de um limite de tempo para a espera de chamadas, em cujo término o processo poderá continuar.

As falhas de comunicação, conforme GENTLEMAN [58], podem advir de causas físicas e lógicas. As falhas provenientes do sistema físico de comunicação, embora raras, são resolvidas através de mecanismos de reconhecimento de mensagens, ou através de limite de tempo ("time-out").

Uma causa bastante comum de falhas lógicas está na presença de impasses ("deadlock"). O impasse poderá resultar de um conjunto de processos que, de forma cíclica, estejam procurando enviar ou receber mensagens de um para outro. A solução desse problema costuma ser obtida por meio da análise de grafos que facilitam a visualização dos impasses. Outra causa de falhas na comunicação deve-se à destruição de processos que estejam envolvidos numa tentativa de comunicação. Isto pode propiciar o aparecimento de execuções indesejáveis denominadas orfãs. Este problema foi tratado por SHRIVASTAVA [156] que utiliza as abordagens de execução *pelo menos uma vez* e *exatamente uma vez* numa estrutura de comunicação do tipo "rendez-vous" estendido ou chamada remota de procedimento. Na abordagem *pelo menos uma vez*, a execução de uma tarefa por uma ou mais vezes leva a um resultado correto, tolerando portanto a existência de orfãs. A abordagem *exatamente uma vez* implica em uma única execução e na intolerância à presença de orfãs. GENTLEMAN [58] ressalta que uma das formas mais sérias de falha de comunicação ocorre quando existem processos desleais, ou seja, quando os processos violam o protocolo padrão de comunicação estabelecido. Este problema é muito sério, quando se trata de processos comunicantes elaborados por programadores diferentes; quando o mesmo programador elaborar to

dos os processos, essa questão poderá ser bastante minimizada.

De qualquer maneira, os sistemas de comunicação devem possuir mecanismos de detecção e de recuperação de falhas, de forma a dar o apoio necessário aos outros níveis de comunicação.

VI.2.6 - USO DE PORTOS ("PORTS")

O porto consiste num elemento do sistema que permite a comunicação entre conjuntos de processos através do mascaramento de seus identificadores. Esta característica, denominada endereçamento funcional, impõe a principal diferença dessa forma de comunicação em relação às outras (VINTER et alii [191]).

Num sistema, tanto os processos quanto os portos deverão ser referenciados através de identificadores globais. De acordo com REID [141] e STEMPLE et alii [174], cada porto deverá ter o seu dono que será o componente que o criar. A criação e interligação dinâmica de portos, estabelecendo os meios necessários para que as mensagens possam fluir pelo sistema com certa flexibilidade, são fatores determinantes na formulação de sistemas que permitem a criação e destruição dinâmicas de processos.

A comunicação entre processos locais ou remotos, num sistema estruturado com o uso de portos, será feita pela execução de primitivas síncronas ou assíncronas do tipo *envia* e *recebe*. A atuação de cada processo restringir-se-á aos portos nele declarados. A interligação dos portos de cada processo com os portos de outros processos estabelecem os trajetos para o fluxo de mensagens entre esses portos.

Os portos têm sido usados em vários sistemas baseados em mensagens como ACCENT (RASHID e ROBERTSON [137]), DEMOS (BASKETT et alii [14]), CLU (LISKOV [103]), etc. O desenvolvimento de um mecanismo de comunicação entre os nós de uma rede, usando a linguagem ADA para a implementação dos portos e para sua criação e manipulação dinâmicas, foi realizado por FANTECHI et alii [52].

VI.2.6.1 - TIPOS DE PORTOS

Os portos são classificados de acordo com as restrições a ele impostas.

Quanto ao sentido de transferência de mensagens e com relação ao processo que o possuir, um porto poderá ser: de entrada, quando usado para receber mensagem; de saída, quando usado para enviar mensagens; e bidirecional ou de difusão, quando a mensagem puder fluir nos dois sentidos. A representação gráfica desses três tipos de portos pode ser vista na figura (VI.4) que também mostra a conexão de vários portos com um único.

VI.2.6.2 - INTERLIGAÇÃO DE PORTOS

A interligação de portos pode dar-se de quatro maneiras: um para um; um para vários; vários para um; e vários para vários.

A conexão *um para um* associa um emissor a um receptor, devendo manter uma fila para que, em caso de necessidade, armazene as mensagens pendentes.

A conexão *um para vários* associa um conjunto de recepto

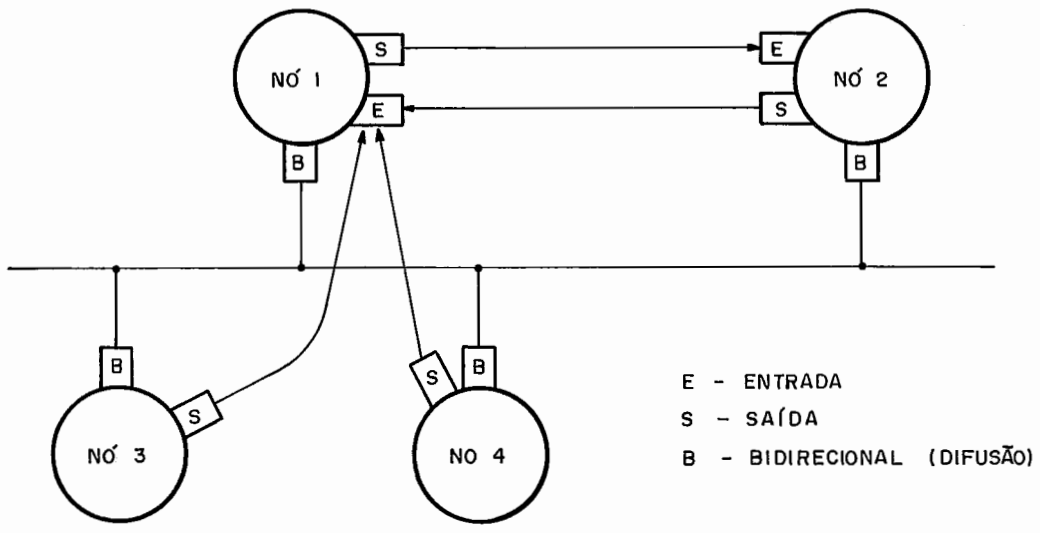


FIGURA VI. 4 - ESQUEMA ILUSTRATIVO DOS TRES TIPOS DE PORTOS

res com um único emissor. Este esquema de comunicação admite as conexões de difusão, onde a mensagem que foi enviada chegará a cada um dos receptores, e as conexões múltiplas de leitura, onde a mensagem enviada só chegará ao primeiro receptor que tentar recebê-la. A conexão múltipla de leitura será bastante útil num ambiente de servidores repetidos que atendam as requisições de serviço conforme a disponibilidade.

A conexão *vários para um* associa vários emissores a um único receptor. Encontram-se aqui os esquemas de conexões múltiplas de escrita e de conexões de concentração. A conexão múltipla de escrita, equivalente à estrutura de endereçamento assimétrico com a primitiva "receive any", faz com que o receptor receba mensagem de qualquer emissor a ele ligado através de portos. A conexão de concentração faz com que a mensagem recebida seja a concatenação de um conjunto de mensagens, provenientes dos emissores ligados ao receptor, de forma que a recepção só estará completa quando todos os emissores tenham enviado mensagens.

A conexão *vários para vários* associa um conjunto de emissores a outro conjunto de receptores. Este tipo de conexão pode ser obtido através da combinação das conexões *um para vários* e *vários para um*, gerando quatro possibilidades a seguir.

a) conexões múltiplas de escrita/conexão de difusão:
   - transmite cada mensagem para todos os receptores do conjunto;

b) conexões múltiplas de escrita/conexões múltiplas de leitura:
   - transmite cada mensagem para o primeiro receptor que quiser recebê-la;

c) conexão de concentração/conexão de difusão:
   - transmite a concatenação das mensagens de todos os emissores do conjunto para todos os receptores do conjunto;

d) conexão de concentração/conexões múltiplas de leitura:
   - transmite a concatenação das mensagens de todos os

emissores para o primeiro receptor que quiser receber-la.

VI.2.6.3 - UTILIZAÇÃO DE "BUFFERS"

A implementação de comunicação baseada no uso de portos requer a capacidade de enfileirar mensagens nos portos à espera de requisições de recepção. A manutenção de filas introduz o problema de alocação de "buffers", cuja solução pode dar-se através de três técnicas: alocação local associada ao porto; alocação local associada ao processo; e alocação global associada ao sistema.

A alocação local associada ao porto consiste na fixação de um espaço de "buffers" para cada porto de entrada. Quando o espaço de "buffers" estiver totalmente preenchido e ocorrer a tentativa de envio de outra mensagem ao porto, poderá acontecer uma das seguintes providências: ampliar de forma controlada o espaço de "buffers"; bloquear o processo emissor até que haja disponibilidade no espaço de "buffers"; retornar ao emissor uma informação de erro.

A alocação local associada ao processo consiste na fixação de espaço de "buffers" para cada processo, limitando a quantidade de mensagens que pode ser recebida em seus portos.

A alocação global associada ao sistema funciona com base na existência de um conjunto de "buffers" de uso comum. Esta técnica poderá gerar problemas de congestionamento no sistema, quando a utilização dos "buffers" atingir o seu limite numérico. A solução desse problema pode ser obtida com a combinação das técnicas de alocação local associada ao porto com alocação global associada ao sistema, assegurando um número de "buffers" exclusivos a cada porto com possibilidade de ampliação com os "buffers" do sistema.

VI.2.6.4 - OPERAÇÕES NECESSÁRIAS AO COMPORTAMENTO DINÂMICO DO SISTEMA

Um sistema apresenta um comportamento dinâmico quando

seus elementos sofrem variações como criação e destruição de processos, criação e destruição de portos, conexão e desconexão de portos, etc.

As operações envolvidas com essas variações, de acordo com REID [141], são as seguintes:

- *criação de processo*: passará a existir um novo processo que começará a execução logo que for criado;

- *destruição de processo*: o processo deixará de existir como parte do sistema;

- *criação de porto*: passará a existir um novo porto, pertencente ao processo que o criar, associado com um identificador definido na operação de criação;

- *destruição de porto*: o porto deixará de existir como parte do sistema. Esta operação só poderá ser executada pelo processo dono do porto;

- *conexão de portos*: será estabelecida a conexão de comunicação entre dois portos. Esta operação poderá ser executada por qualquer processo que conhecer os dois portos. Os portos só poderão ser conectados se forem compatíveis e de sentidos opostos. Um porto poderá estar ligado a muitos portos ao mesmo tempo e a tentativa de conexão de portos já ligados não terá nenhum efeito;

- *desconexão de portos*: será desfeita a conexão de comunicação entre dois portos. Esta operação poderá ser executada por qualquer processo que conhecer os dois portos envolvidos.

Para que essas operações possam ser executadas corretamente sem a possibilidade de conflitos, deve-se satisfazer algumas restrições na sequência de operações: um porto deverá ser criado antes do estabelecimento de suas conexões; as conexões deverão ser realizadas antes da aplicação de primitivas de comunicação sobre eles; um porto deverá ser desconectado completamente antes de ser destruído; o processo possuidor do porto deverá

ser criado antes da criação do porto; o porto deverá ser destruído antes da destruição do processo que o contiver.

VI.2.6.5 - OUTRAS CONSIDERAÇÕES SOBRE PORTOS

Além do que foi considerado, o tratamento de portos envolve outras questões como transferência de direito de propriedade, transferência de direito de acesso, estabelecimento de mecanismos de proteção baseados em portos, etc. A abordagem desses e de outros assuntos tem sido feita na literatura já mencionada neste capítulo e em outras referências, entre as quais aquelas de autoria de BERRY et alii [16], MAO e YEH [111], RASHID [136], JOSEPH [77], SILBERSCHATZ [160], BOCHMANN e RAYAL [19], e WILLIAMSON e HOROWITZ [199].

C A P Í T U L O VII

PROJETO DE SUBSISTEMAS DE COMUNICAÇÃO

A atividade de pesquisar e desenvolver sistemas distribuídos requer a presença de uma infraestrutura adequada de hardware e software, constituída por um conjunto de processadores operadores interligados através de um subsistema de comunicação, e de um sistema operacional apropriado. Nesse contexto, o subsistema de comunicação apresenta-se como um elemento fundamental, cujo projeto exerce permanente influência no comportamento e desempenho do sistema.

Visando dispor de ambientes que permitissem o estudo e desenvolvimento de sistemas distribuídos, optou-se, nesse trabalho, pela implementação de dois tipos diferentes de subsistemas de comunicação para servirem como bancadas de trabalho. Nesse sentido, os artigos de BRAYER e LAFLEUR [20], LINDSAY [102], e TOKUDA e MANNING [188] foram determinantes para a escolha dos subsistemas de comunicação baseados em conexão ponto a ponto e em barramento paralelo centralizado.

Dessa maneira, a abordagem das questões de hardware e de software de subsistemas de comunicação, utilizando canais de conexão ponto a ponto e canais de difusão, permitiria que a maior parte dos problemas referentes ao suporte básico de sistemas distribuídos fosse analisada. Além disso, os dois tipos de subsistema de comunicação possibilitariam que fosse feita uma análise comparativa da implementação dos principais mecanismos de comunicação, bem como de suas variações, mostrando sua aderência ou dificuldade em cada caso.

Pelo fato de ter prevalecido o interesse acadêmico, embora não se tenha perdido de vista as aplicações profissionais, o enfoque principal no desenvolvimento dos dois tipos de subsistema de comunicação concentrou-se na análise de comportamento, ficando em segundo plano as questões de desempenho. Por isso,

deu-se grande importância à flexibilidade e facilidade de manipulação dos circuitos, o que indicou nos dois casos o uso de componentes da família de microprocessadores para a elaboração dos projetos. Apesar dessa estratégia ter sido utilizada, o desenvolvimento dos projetos transcorreu o tempo todo com a preocupação de identificação dos pontos de alteração que, às custas de maior custo, complexidade, etc., poderiam refletir numa melhora substancial de desempenho.

Nos dois tipos de subsistema de comunicação, o tratamento de falhas apresenta-se como uma questão de grande importância nos vários níveis de projeto. No caso do projeto do subsistema de comunicação ponto a ponto, esse tratamento é realizado principalmente a nível de software. No caso do projeto do subsistema de comunicação baseado no barramento paralelo centralizado, o tratamento de falhas é feito de forma equilibrada a nível de hardware e de software.

VII.1 - SUBSISTEMA DE COMUNICAÇÃO COM CONEXÃO PONTO A PONTO

VII.1.1 - ESTRUTURA E ASPECTOS DO HARDWARE DO SUBSISTEMA

O subsistema de comunicação com conexão ponto a ponto baseia-se na existência de vários processadores de comunicação, cada um com alguns canais de comunicação ponto a ponto, interligados de alguma maneira. Esses processadores, manipulando pacotes de informação e funcionando com a técnica armazena e envia, estabelecem os trajetos para a transferência de mensagens de um processador operador para outro.

Esta estrutura de subsistema de comunicação possibilita a abordagem de questões ligadas com roteamento de informações, congestionamento, controle de fluxo, etc., além de permitir o uso de estratégias de reconfiguração de sistemas para contornar falhas ou elaborar sistemas dinâmicos.

Para efeito de concepção do processador de comunicação, optou-se por sua inclusão num subsistema de comunicação com conexão ponto a ponto em topologia irregular, conforme a figura

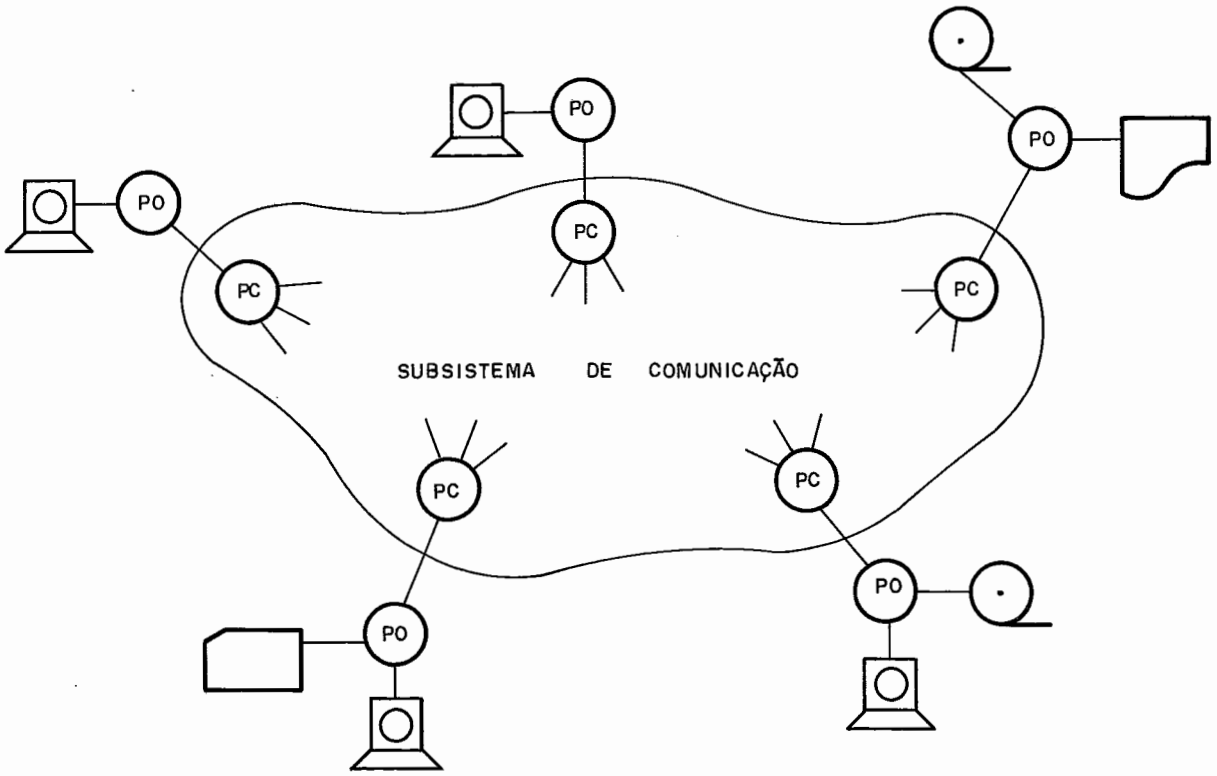


FIGURA VII.1 - CONFIGURAÇÃO GENÉRICA DO SISTEMA DISTRIBUÍDO COM CONEXÃO PONTO A PONTO

(VII.1). Optou-se também por conectá-lo a um nó operador e a três processadores de comunicação vizinhos, usando para isso canais seriais "full duplex".

Cabe citar que as comunicações internas ao subsistema de comunicação foram estruturadas para usarem linhas seriais compatíveis com o padrão de comunicação EIA RS-422 (MCNAMARA [116]), funcionando a nível de enlace de dados com o protocolo HDLC simplificado. A comunicação do processador de comunicação com o processador operador foi prevista para usar linhas seriais compatíveis com o padrão RS-422 ou RS-232C, visando facilitar a ligação com várias máquinas comerciais, funcionando com o protocolo HDLC ou algum outro.

A elaboração do processador de comunicação, conforme pode ser visto na figura (VII.2), foi baseada no uso do microprocessador Z80-A e de circuitos integrados da mesma família ou com ela compatíveis (KIRNER [84]). O circuito detalhado do processa

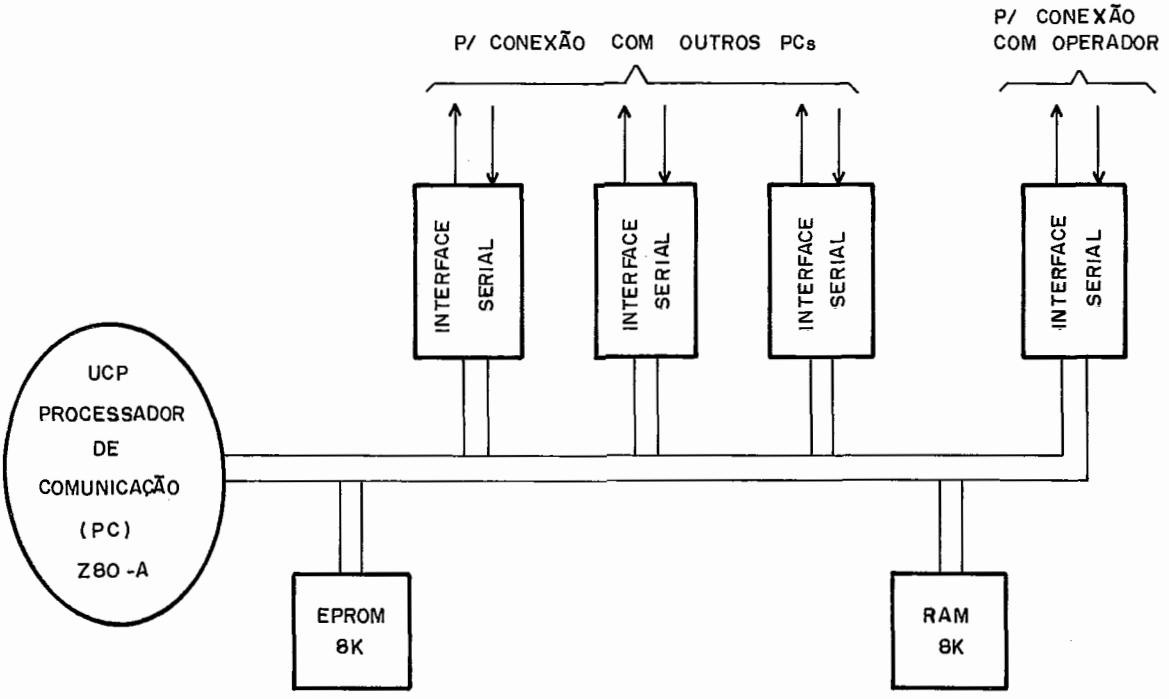


FIGURA VII. 2 - CONFIGURAÇÃO DO PROCESSADOR DE COMUNICAÇÃO (PC) PARA CONEXÃO PONTO A PONTO

dor de comunicação para conexão ponto a ponto consta no apêndice A.

As interfaces seriais foram implementadas com a utilização do circuito integrado Z80-SIO ("Serial Input Output"), funcionando em regime de comunicação síncrona a uma taxa de transmissão da ordem de 20 K bits/seg, limitada à capacidade de atendimento simultâneo do processador de comunicação (TEIXEIRA [180]). A conexão serial do processador de comunicação com o processador operador (PC-PO) apresenta a alternativa de uso do circuito 8251, ou equivalente, funcionando em regime de comunicação assíncrona a uma taxa de transmissão selecionável de até 9600 bits/seg.

A alternativa de conexão serial (PC-PO) de baixa velocidade levou em conta vários fatores, dentre eles: a disponibilidade de interfaces seriais deste tipo em máquinas comerciais; a facilidade de ligação e de uso; e a viabilidade de aplicação em situações de transferências eventuais de informação. Para os casos de transferência sistemática de grande quantidade de informação, pode-se utilizar a conexão serial (PC-PO) de alta velocidade, ou

outras formas de conexão de maior velocidade de transmissão como linhas paralelas ou barramento comum. O inconveniente desses esquemas alternativos esbarra muitas vezes na necessidade de adaptações de hardware e software no processador operador, além de exigir que tais esquemas existam no processador de comunicação.

As interfaces seriais do processador de comunicação implementadas com SIO têm estrutura para fazerem transmissão e recepção por interrupção, de tal modo que, dependendo do software e das limitações de tempo ou de velocidade, se possa executar essas operações em paralelo.

VII.1.2 - ASPECTOS DE SOFTWARE

O software básico do sistema distribuído consiste do agrupamento do software de cada processador de comunicação, devidamente complementado com a parcela de software do processador operador necessária para a comunicação (PC-PO).

O software do processador de comunicação envolve o programa supervisor, rotinas de comunicação pelas linhas seriais do subsistema de comunicação (MARQUES [112]), rotinas de comunicação pela linha serial conectada ao processador operador, e rotinas de suporte para a implementação das primitivas do sistema e do protocolo de comunicação.

O software do processador operador, além do software próprio da máquina, inclui rotinas de comunicação pela linha serial conectada ao processador de comunicação, e rotinas referentes às primitivas de comunicação do sistema distribuído.

Como as primitivas do sistema distribuído devem atuar ao nível do processador operador e ao nível do subsistema de comunicação, a implementação de cada primitiva depende de rotinas complementares existentes nesses dois níveis. As primitivas são implementadas no núcleo do sistema, conforme a figura (VII.3).

Num nível mais alto, pode-se encontrar o sistema operacional distribuído, fazendo com que os detalhes do subsistema de comunicação sejam transparentes, ou algum programa aplicativo simples, utilizando a infraestrutura de interconexão existente como, por exemplo, um sistema servidor de periféricos ou um sistema de correio eletrônico.

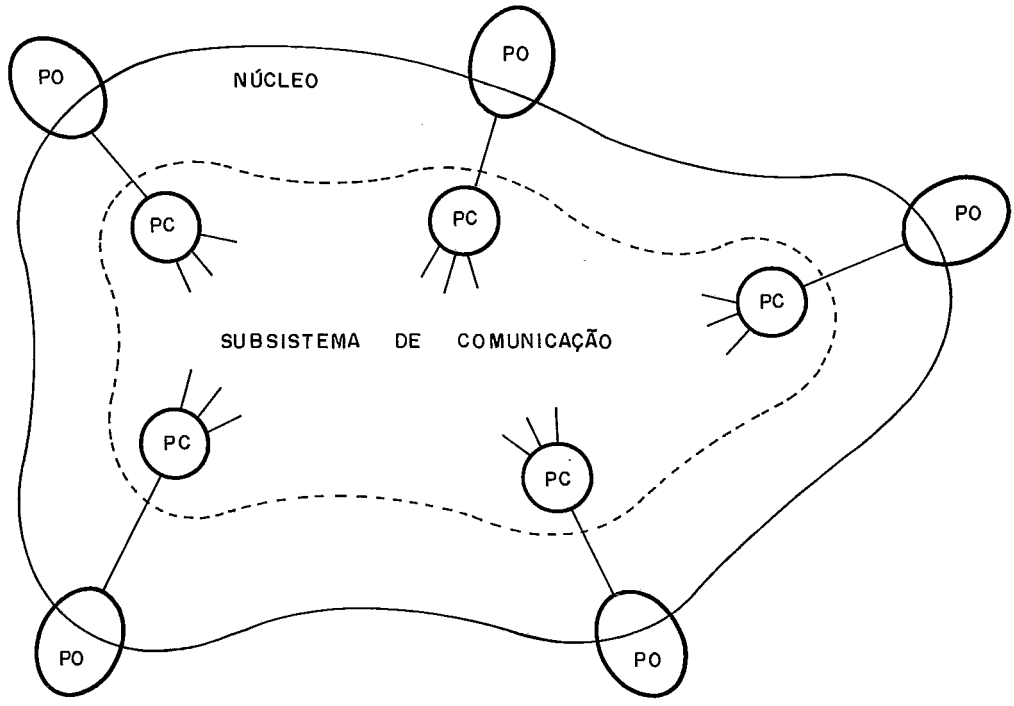


FIGURA VII.3 - NÚCLEO DO SISTEMA DISTRIBUÍDO COM CONEXÃO PONTO A PONTO

No processador de comunicação, o programa supervisor é o responsável pelo controle de fluxo de informações no subsistema de comunicação, cuidando do estabelecimento de rotas para os pacotes de mensagens, assim como a manipulação das filas (KIRNER [84], MORON [118], SILVA [161]). Para isto, de acordo com a figura (VII.4), o supervisor conta com quatro pares de filas de transmissão, cada um associado a uma linha serial, e com quatro "buffers" igualmente associados. A existência de um par de filas de transmissão para cada linha serial é justificável para otimizar os espaços de memória e agilizar o protocolo de comunicação, pois enquanto uma fila se presta para guardar pacotes longos de informação, a outra serve para pacotes curtos de controle.

No caso de recepção, a cada "byte" recebido em qualquer das linhas, ocorrerá interrupção e a rotina de recepção irá preenchendo o "buffer" correspondente até acabar, quando então haverá uma anotação de "buffer" com recepção terminada. Por outro lado, a transmissão de pacotes de mensagem será feita "byte" a "byte" numa única linha por vez até terminar.

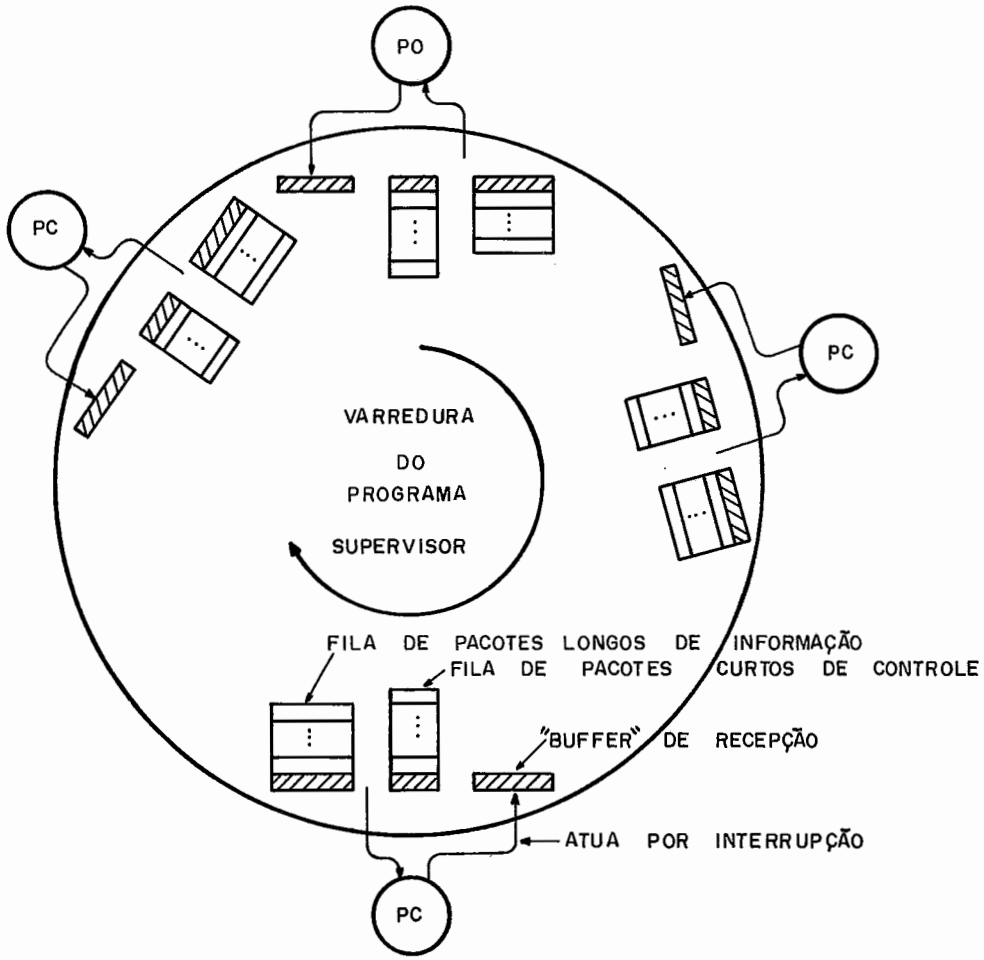


FIGURA VII. 4 - ESTRUTURA DE FILAS E "BUFFERS" NO PROCESSADOR DE COMUNICAÇÃO PARA CONEXÃO PONTO A PONTO

O programa supervisor deverá passar alternadamente pelas etapas de recepção e transmissão. Na etapa de recepção, o programa supervisor fará a varredura de todos os "buffers", verificando se a recepção já está terminada. Para cada "buffer" com recepção terminada, será feita a coleta do pacote, determinação da rota, e colocação do conteúdo do "buffer" na fila apropriada. Na etapa de transmissão, o programa supervisor, a partir do último par de filas atendido na etapa de transmissão anterior, pegará um pacote de uma das filas do próximo par de filas, conforme uma ordem pré-estabelecida, e comandará sua transmissão pela linha serial correspondente. A escolha do pacote de uma das filas de um par de filas dar-se-á com prioridade para a fila de paco

tes curtos de controle.

O protocolo HDLC simplificado utilizado entre os processadores de comunicação é baseado na troca de pacotes curtos de controle, visando indicar reconhecimento positivo com autorização de nova transmissão, reconhecimento positivo sem autorização de nova transmissão, reconhecimento negativo, etc. O aspecto fundamental do protocolo é que deve existir uma autorização do receptor para que a transferência ocorra. A condição inicial do sistema prevê a existência prévia de autorização em todas as linhas.

As primitivas de comunicação implementadas no núcleo serão responsáveis pela comunicação entre processadores operadores. O detalhamento dessas primitivas, projetadas para atuar numa configuração específica de rede em laço com detecção e recuperação automática de falhas, será visto no próximo capítulo.

VII.1.3 - CRÍTICAS E ALTERNATIVAS DE PROJETO

Devido ao fato das recepções ocorrerem por interrupção, a taxa de transmissão nas linhas seriais tem limitação máxima imposta pela capacidade de atendimento, no pior caso, de quatro interrupções de recepção e uma de transmissão, no intervalo entre a chegada ou saída de dois "bytes" consecutivos numa mesma linha. Apesar dessa solução ser bastante simples, e evitar a perda de informações, ela apresenta o inconveniente de fazer com que o subsistema de comunicação funcione com uma taxa de transmissão mínima para qualquer carga de trabalho.

Uma solução alternativa que procurasse aumentar a taxa de transmissão nas linhas de comunicação poderia ser obtida às custas da ocorrência eventual de retransmissão de informação. Esta solução consistiria em dimensionar a taxa de transmissão para o melhor caso, ou seja, para o atendimento de uma única linha por vez. Isto implicaria em ignorar ou descarregar as informações de outras linhas toda vez que aparecessem simultaneamente com a que estivesse sendo atendida. Tal alternativa, embora propiciasse um melhor desempenho com cargas baixas, acabaria deteriorando consideravelmente o desempenho em situações de cargas altas, devido à necessidade de inúmeras retransmissões até que

um sucesso fosse obtido. O dimensionamento da taxa de transmissão das linhas de comunicação para uma situação de carga normal, configurando uma solução intermediária, minimizaria a necessidade de retransmissão de pacotes, embora não a eliminasse.

Uma alternativa para melhorar o desempenho do subsistema de comunicação, sem precisar realizar eventuais retransmissões, consistiria em utilizar circuitos de acesso direto à memória nas interfaces do processador de comunicação com as linhas, funcionando com uma taxa de transmissão dimensionada para o pior caso.

A conexão ponto a ponto do sistema adotado apresenta algumas vantagens como facilidade para contornar falhas, possibilidade de reconfiguração dinâmica, paralelismo de fluxo de informação, etc. O paralelismo de fluxo de informação obtido pela presença de várias linhas funcionando ao mesmo tempo exerce alguma compensação no desempenho do subsistema de comunicação.

Por outro lado, esse tipo de subsistema de comunicação apresenta problemas críticos do ponto de vista de implementação de redes locais e multiprocessadores. Dentre esses problemas encontram-se: a elevada limitação na taxa de transmissão das linhas seriais; o tempo de atraso relativamente alto em transferências que envolvam alguns processadores de comunicação intermediários; a dificuldade de implementação de algumas formas de comunicação importantes, como a transmissão dos tipos *um para vários* e difusão ("broadcasting"); etc. Esses problemas aparecem, em grande parte, devido à presença de distâncias variáveis e filas no trajeto entre o nó origem e o nó destino.

VII.2 - SUBSISTEMA DE COMUNICAÇÃO COM PARRAMENTO PARALELO CENTRALIZADO

No desenvolvimento desse tipo de subsistema de comunicação, procurou-se explorar as vantagens da interconexão de processadores operadores por meio de canais de difusão. Ao mesmo tempo, procurou-se contornar suas deficiências, visando a obtenção de uma infraestrutura apropriada ao estudo de primitivas de comunicação para sistemas de aplicação em tempo real, e à elaboração

de sistemas operacionais distribuídos.

Nesse sentido, o subsistema de comunicação deveria apresentar alguns atributos específicos tais como: alta velocidade; alta confiabilidade; tempo máximo bem definido de comunicação entre dois elementos da rede; e custo não proibitivo.

Para obter-se um tipo de rede, que possuisse todas essas qualidades, foi feito o seguinte:

- tomou-se o barramento paralelo, que apresenta alta velocidade e baixa confiabilidade, e introduziu-se linhas e circuitos redundantes, dotando-o com tolerância a falhas para torná-lo altamente confiável;

- em seguida, escolheu-se um método determinístico de acesso ao barramento baseado em disputa de prioridade com único vencedor (esquema da reserva);

- implementou-se o método de acesso ao barramento, usando-se um hardware distribuído, e obtendo-se um tempo máximo de comunicação bem definido e pequeno para o pior caso;

- para conseguir-se custos não proibitivos, optou-se pelo uso de componentes da família de microprocessadores e pelo uso de linhas de transmissão de velocidade média.

Da mesma maneira que no projeto anterior, optou-se pela utilização de componentes da família de microprocessadores nos pontos possíveis, aproveitando-se as vantagens de custo e flexibilidade desses componentes. No entanto, tomou-se todo o cuidado para que a estruturação do projeto permitisse a utilização alternativa de estruturas mais rápidas e mais adequadas a determinadas exigências de aplicações especiais.

O esquema de barramento centralizado foi discutido por LINDSAY [102] e ACAMPORA e HLUCHYJ [1], tendo sido utilizado neste trabalho com o enfoque distribuído, conforme KIRNER e MARQUES [85].

VII.2.1 - ARQUITETURA E FUNCIONAMENTO DO SUBSISTEMA

O subsistema de comunicação com barramento paralelo centralizado constitui-se de vários processadores de comunicação (PC) interligados através de um barramento paralelo disposto num painel traseiro, conforme a figura (VII.5).

O subsistema de comunicação é montado em um gabinete, contendo uma fonte confiável ("nobreak"), o barramento paralelo com 72 linhas dispostas no painel traseiro, e 64 placas de circuito ligadas ao barramento, cada uma correspondendo a um processador de comunicação. Este conjunto poderá ser colocado dentro de uma sala bem protegida, de onde sairão todas as ligações para os processadores operadores remotos e eventualmente para dispositivos periféricos, sensores, e atuadores, que estiverem ligados a processadores de comunicação expandidos. Cabe salientar que um processador de comunicação expandido contém, além das funções

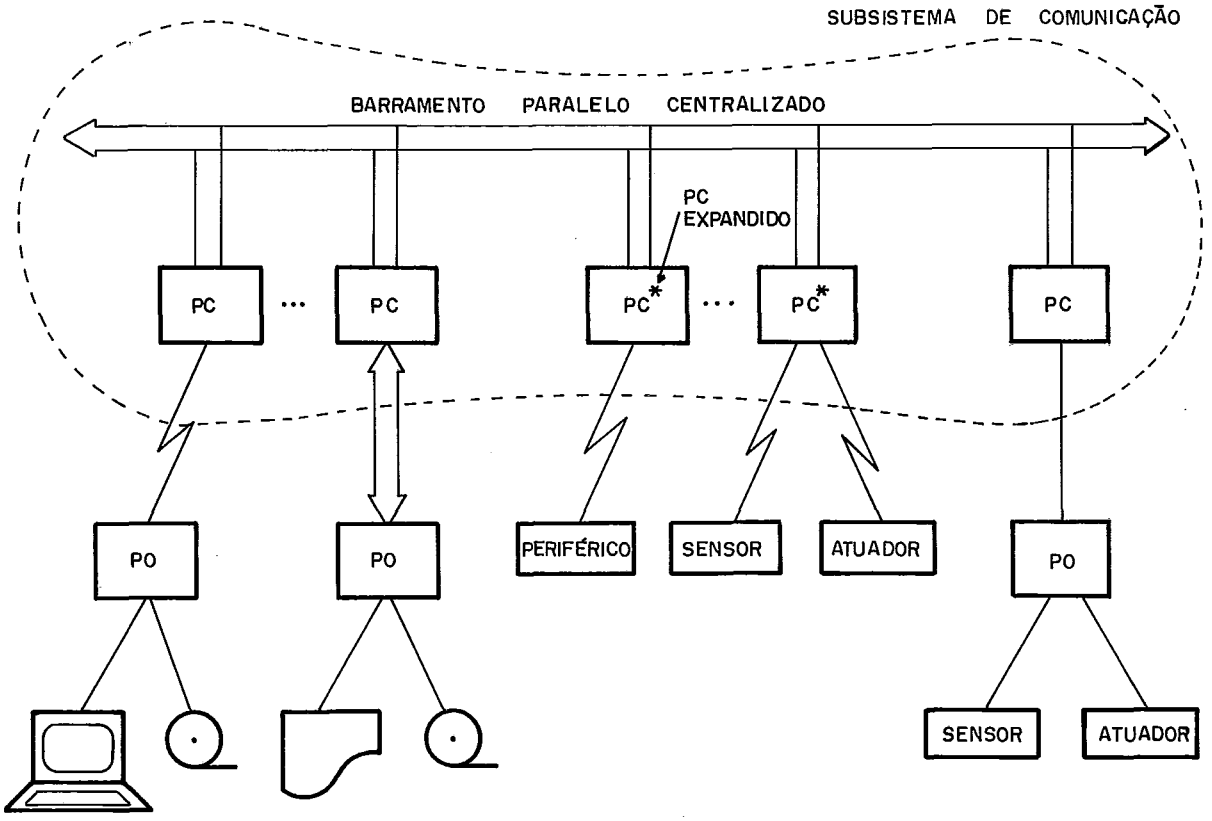


FIGURA VII.5 - CONFIGURAÇÃO GENÉRICA DO SISTEMA DISTRIBUÍDO COM BARRAMENTO PARALELO CENTRALIZADO

de um processador de comunicação normal, funções simples de controle de alguns periféricos, tais como terminal de vídeo e impressora, e funções de leitura de dispositivos sensores e de acionamento de dispositivos atuadores. A limitação a 64 processadores de comunicação num barramento deve-se às restrições de componentes eletrônicos "tri-state" e à necessidade de manter-se o comprimento do barramento a dimensões reduzidas. A centralização do subsistema de comunicação, além de facilitar a manutenção, contribui bastante para o aumento da segurança do sistema, uma vez que os maiores riscos de ruptura, incidindo com maior frequência nos cabos externos, comprometerão somente a comunicação individual de cada processador operador com o subsistema, sem afetar sensivelmente o funcionamento do conjunto.

A comunicação entre processadores operadores ocorrerá através dos processadores de comunicação correspondentes, e do barramento paralelo centralizado de alta velocidade que constitui o meio compartilhado de comunicação. A alta taxa de transferência de informação entre processadores de comunicação é obtida muito mais pela existência de várias linhas no barramento paralelo, do que pela velocidade de cada linha. Para permitir que todos os processadores de comunicação utilizem o barramento, é necessário que cada um utilize-o no menor tempo possível, sem monopolizá-lo. Por isso, cada processador deverá contar com "buffers" de emissão e "buffers" de recepção, conforme a figura (VII.6). Esses "buffers" servirão para a montagem da informação a ser transferida, e para a desmontagem da informação recebida, de forma a ser utilizada no processador de comunicação ou retransmitida em taxas menores para o processador operador correspondente.

Um processador operador emissor fará a transferência de informação para um processador operador receptor da seguinte forma. Primeiro, o processador operador emissor transferirá a informação para o processador de comunicação origem a ele ligado, por "bit" ou por "byte", dependendo da ligação ser serial ou paralela. Esse processador de comunicação irá coletando as partes da informação, conforme a sua chegada, e montando-as convenientemente no "buffer" de emissão. Quando a montagem terminar, o processador de comunicação origem acionará o seu circuito de disputa do barramento e, tão logo ganhe acesso, verificará se o pro

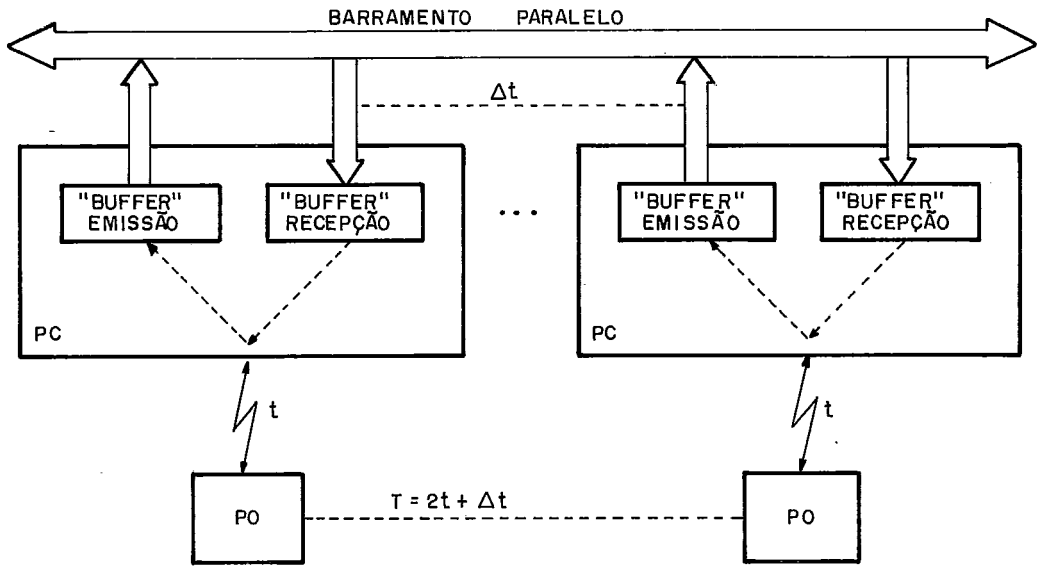


FIGURA VII. 6 - COMUNICAÇÃO ENTRE 2 PROCESSADORES OPERADORES NUM SISTEMA COM BARRAMENTO

cessador de comunicação destino está pronto para receber a informação ("buffer" de recepção vazio). No caso de processador de comunicação destino não encontrar-se pronto, o processador de comunicação origem desistirá do barramento e aguardará um tempo para nova tentativa. No caso do processador de comunicação destino estar pronto, então haverá a transferência da informação do "buffer" de emissão do processador de comunicação origem para o "buffer" de recepção do processador de comunicação destino. Este último processador encarregar-se-á de desmontar a informação recebida e de transferí-la ao processador operador numa taxa de transmissão compatível com a ligação.

O tempo máximo de comunicação entre dois processadores operadores no pior caso, ou seja, quando n processadores estiverem competindo pelo barramento, e esses dois forem atendidos por último, deverá ser a soma dos seguintes ítens:

- tempo gasto na transmissão PO-PC e no preenchimento do "buffer" de emissão;

- n vezes o tempo de transferência de um "buffer" pelo barramento paralelo, correspondendo ao tempo de espera mais o tempo de transferência;

- tempo gasto no esvaziamento do "buffer" receptor e respectiva transmissão PC-PO.

Como o tempo de transferência de um "buffer" pelo barramento deverá ser da ordem de centenas ou milhares de vezes menor do que o tempo gasto na transmissão PO-PC e no preenchimento do "buffer" ou vice-versa, ele terá uma influência muito pequena no tempo total de comunicação entre processadores operadores. Isto ocorrerá sempre que for assegurado que os processadores de comunicação estejam prontos para recepção. Sabendo-se que esta condição muitas vezes não corresponde à realidade, a característica de tempo bem definido de comunicação desaparecerá, a não ser que sejam tomadas providências que contornem a possibilidade de existir tentativas de transferência de "buffer" para um processador de comunicação ocupado. Uma dessas providências consistiria na introdução de "buffers" adicionais em cada processador de comunicação, e em número suficiente para armazenar todas as informações que pudessem ser transferidas para esse processador dentro de períodos curtos de tempo. Além do armazenamento, esta providência permitiria o tratamento dos "buffers" de acordo com uma ordem de prioridade. Nessa situação, o tempo máximo de comunicação entre dois processadores operadores poderia variar, conforme as prioridades envolvidas, mas dentro de valores perfeitamente previsíveis.

É interessante notar em toda essa análise que é impossível que um processador de comunicação monopolize o uso do barramento, ou impeça a transferência de informação de processadores de comunicação de baixa prioridade, ao tentar transferir sequências de informação. Isto se deve ao fato do preenchimento de "buffer", que é uma atividade bem mais demorada do que a transferência pelo barramento, propiciar o intervalo de tempo necessário para as outras transferências pendentes.

VII.2.2 - PROJETO DO BARRAMENTO PARALELO

O barramento paralelo foi concebido inicialmente para ter 72 linhas, sendo 40 para transferência de informação (pacotes de mensagem), 4 para atuar como reserva das linhas de infor

mação, 8 para o endereço origem, 8 para o endereço destino, e as 12 restantes para controle, conforme o quadro (VII.1).

As linhas de controle, além de indicar o estado do barramento, servem também para implementar o protocolo necessário à transferência de informações, para alterar o padrão de transferência, e para realizar testes e ajustes do subsistema de comunicação. O estado do barramento poderá ser verificado através das linhas *desocupado* e *erro-tensão*. O protocolo de transferência utilizará as linhas *limpo-para-envio*, *não-autorizo* e *clock-in*. O padrão de transferência da parte útil do pacote poderá ser de 32 "bits", com uso pleno do barramento, ou de 8 "bits", quando houver necessidade de transmissão mais rápida de pacotes menores, como acontece com pacotes de controle. O uso do barramento no padrão de 8 "bits" também será necessário, quando aparecerem algumas falhas não contornáveis em transmissão com 32 "bits". A alteração de padrão de transferência utilizará as linhas *modo-transmissão* e *reformulação*. A realização de testes e ajustamento do subsistema de comunicação será feita através das linhas *reset*, *teste*, e *desconectar*.

Com o barramento, funcionando a uma taxa da ordem de 5 Mega pacotes/seg, levando em conta as atividades de requisição do barramento, transferência propriamente dita, e liberação do barramento, em cada transferência, e considerando que cada pacote possui 40 "bits", o barramento paralelo proposto corresponderia a um barramento serial funcionando na faixa de 200 Mega "bits"/seg. Desta maneira, através da ampliação do número de linhas de dados, poder-se-ia chegar de forma viável a um barramento paralelo que trabalhasse com uma taxa de transferência correspondente a Giga "bits"/seg em um barramento serial.

VII.2.3 - PROJETO DO PROCESSADOR DE COMUNICAÇÃO

VII.2.3.1 - DESCRIÇÃO DO PROCESSADOR DE COMUNICAÇÃO

O processador de comunicação é o elemento responsável pela conexão do usuário (processador operador) à rede. Cabe a ele controlar todo o fluxo de informação entre ele e o processa

Nº DE LINHAS	NOME	
32	dados	- transferir a informação útil de um pacote.
8	cabeçalho	- transferir o cabeçalho de um pacote.
4	reserva	- servir de linhas sobressalentes para as linhas de transferência de um pacote.
8	endereço origem	- permitir a disputa pelo acesso ao barramento e indicar a origem do pacote a ser transferido.
8	endereço destino	- indicar qual processador de comunicação irá receber o pacote.
1	desocupado	- indicar se o barramento está ocioso.
1	$\overline{\text{clock-in}}$	- disparar o carregamento do pacote no processador de comunicação destino.
1	$\overline{\text{reset}}$	- colocar todos os processadores de comunicação na condição inicial.
1	$\overline{\text{desconectar}}$	- isolar processador de comunicação com falha.
1	$\overline{\text{erro-tensão}}$	- indicar tensão no barramento fora de especificação.
1	$\overline{\text{limpo-para envio}}$	- indicar que o processador de comunicação destino está pronto para receber o pacote.
1	$\overline{\text{teste}}$	- solicitar a execução das rotinas de teste.
1	modo-transm.	- indicar transferência de 8 ou 32 bits.
2	reformul.	- indicar qual dos 4 "bytes" das linhas de dos estará sendo usado.
1	$\overline{\text{não-autor.}}$	- indicar que o processador de comunicação destino não está pronto para recepção.
1	paridade	- indicar a paridade do pacote.
72	barramento	- total das linhas

QUADRO VII.1 - DETALHAMENTO DAS LINHAS DO BARRAMENTO PARALELO

dor operador e vice-versa, e, dentre as suas diversas tarefas, destacam-se as seguintes:

- comunicar-se com o processador operador, recebendo e enviando informação;
- manipular os "buffers", colocando e retirando informação;
- controlar o acesso ao barramento;
- estabelecer a comunicação com outro processador de comunicação, cuidando do protocolo de transferência de informação.

O processador de comunicação é formado por três módulos, consistindo de: interface com o barramento; interface com o processador operador; e controle das interfaces, de acordo com a figura (VII.7). Os circuitos desses módulos encontram-se no apêndice B.

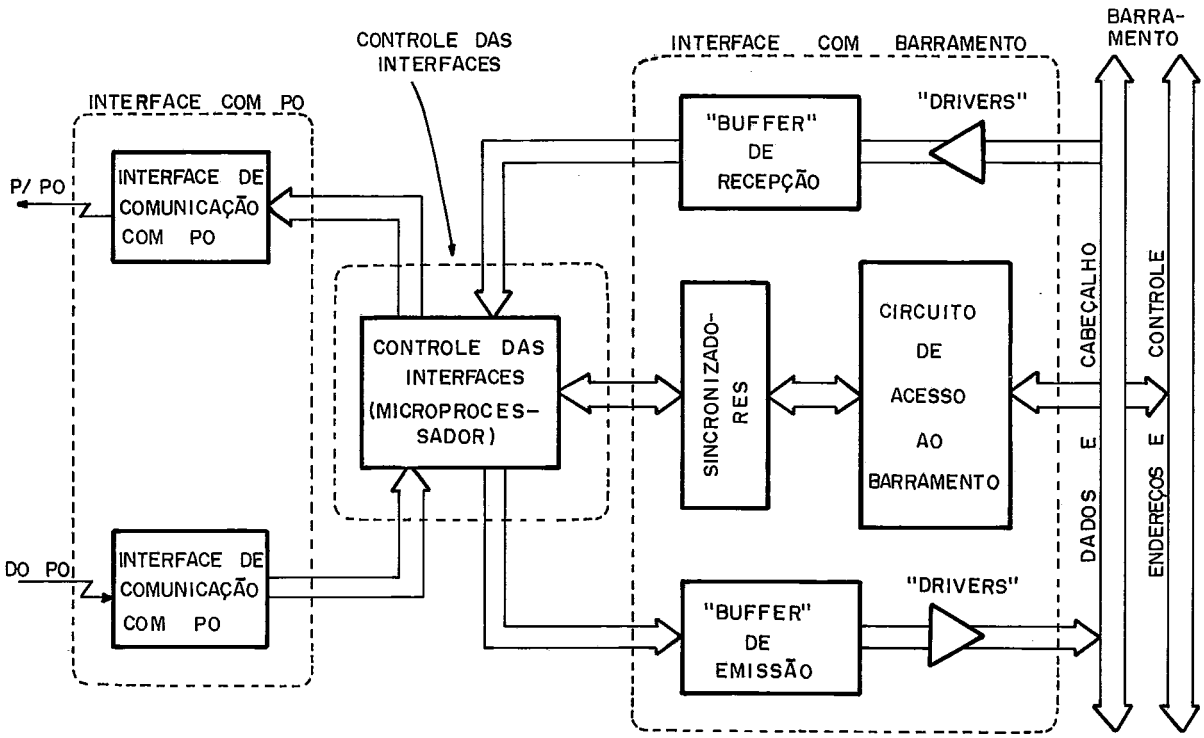


FIGURA VII.7 - CONFIGURAÇÃO DO PROCESSADOR DE COMUNICAÇÃO PARA CONEXÃO COM BARRAMENTO

O módulo de interface com o barramento é constituído pelos seguintes elementos: circuito de acesso ao barramento; sincronizadores; e "buffers" de emissão e de recepção.

O módulo de interface com o processador operador consiste de interfaces padrão de entrada e saída de microprocessadores (8251, SIO, etc.) ou de circuitos especiais. A conexão com o processador operador pode assumir muitas variações: as ligações podem ser feitas com o uso de fios ou fibras óticas, de maneira serial ou paralela, com velocidade baixa ou alta, e, quando não se utilizar processador operador devido à expansão do processador de comunicação, essas linhas poderão estar ligadas a periféricos, sensores, atuadores, e outros elementos.

O módulo de controle das interfaces consiste de circuitos de microprocessador, ou de um hardware especial equivalente. Sua função resume-se na coordenação dos trabalhos das interfaces, e na transferência de informação entre elas.

Quando a conexão do processador de comunicação com o processador operador, ou elemento correspondente, exigir velocidades mais altas, não suportadas pelos microprocessadores e suas interfaces, os módulos controle das interfaces e interface com o processador operador deverão ser construídos com técnicas e componentes adequados às exigências.

Dos elementos do módulo de interface com o barramento, o circuito de acesso é quem se encarregará da disputa pelo barramento. Ele decidirá por meio de prioridade se o processador de comunicação, no qual está incluído, terá conseguido ou não o direito de acessar o barramento naquele instante. A prioridade de cada processador de comunicação será baseada no seu endereço origem diferente de todos os outros, o que garante que a disputa terá sempre um único vencedor. Os sincronizadores, por sua vez, são os elementos que retardam ou armazenam sinais necessários à sequencialização das ações de disputa, uso, e liberação do barramento. Os "buffers" são registradores de dados que colocam ou retiram informação do barramento em determinados instantes definidos pelos sincronizadores. Os "drives", que estão entre os "buffers" e o barramento, e que também são usados em outras linhas, são circuitos "tri-state". No primeiro caso, eles funcionam como isoladores, mas no circuito de disputa pelo barramento eles implementam lógica de fiação.

VII.2.3.2 - CIRCUITO DE ACESSO AO BARRAMENTO

O circuito de acesso ao barramento existente em cada processador de comunicação, conforme a figura (VII.8), é o elemento que permite que, dentre os processadores que estiverem concorrendo pelo acesso ao barramento, o de maior prioridade saia vencedor e ganhe o direito de usar o barramento por um ciclo. Este circuito é um aperfeiçoamento daqueles apresentados por CORSO e VERRUA [38] e TEIXEIRA e TREVELIN [181]. O diagrama de estados mostrado na figura (VII.9) é um elemento complementar para o entendimento do funcionamento do circuito de acesso ao barramento. Enquanto o estado "liberado para disputa" é um estado de repouso, os outros estados são temporários, uma vez que, depois de um tempo bem definido, ou aparecerá uma transição natural pela ocorrência de alguma entrada, ou deverá haver uma transição forçada para o estado de repouso. Isto deverá acontecer para que a ocorrência de falha, tentando fixar o circuito num estado temporário, não possa comprometer todo o sistema.

O circuito de acesso ao barramento está projetado para ser acoplado num barramento com até 64 processadores de comunicação e funciona da seguinte maneira. Quando um processador de comunicação desejar o domínio do barramento, ele deverá manifestar-se através do sinal *quero*. Se o barramento estiver ocupado (*desocupado*=0), o pedido ficará pendente até o mesmo desocupar. No caso do barramento estar desocupado (*desocupado*=1), ou ser desocupado com pedido pendente, então todos os processadores de comunicação que estiverem pedindo acesso naquele instante iniciarão simultaneamente uma disputa, e inibirão a entrada de outros, indicando que o barramento estará ocupado (*desocupado*=0). Aqueles que conseguirem passar colocarão seus endereços (prioridades) no barramento de endereço origem e, através da lógica por fiação com componentes "tri-state", será feita a comparação simultânea dos "bits" de endereço (o endereço mais alto será prioritário). Os circuitos que perderem a disputa deverão esperar a liberação do barramento para participarem da nova disputa (o sinal *quero* de cada um permanecerá). Nessa nova disputa, além dos perdedores das disputas anteriores, poderão aparecer novos competidores. O circuito que ganhar a disputa colocará o endereço destino no barramento e ficará esperando uma resposta para reti

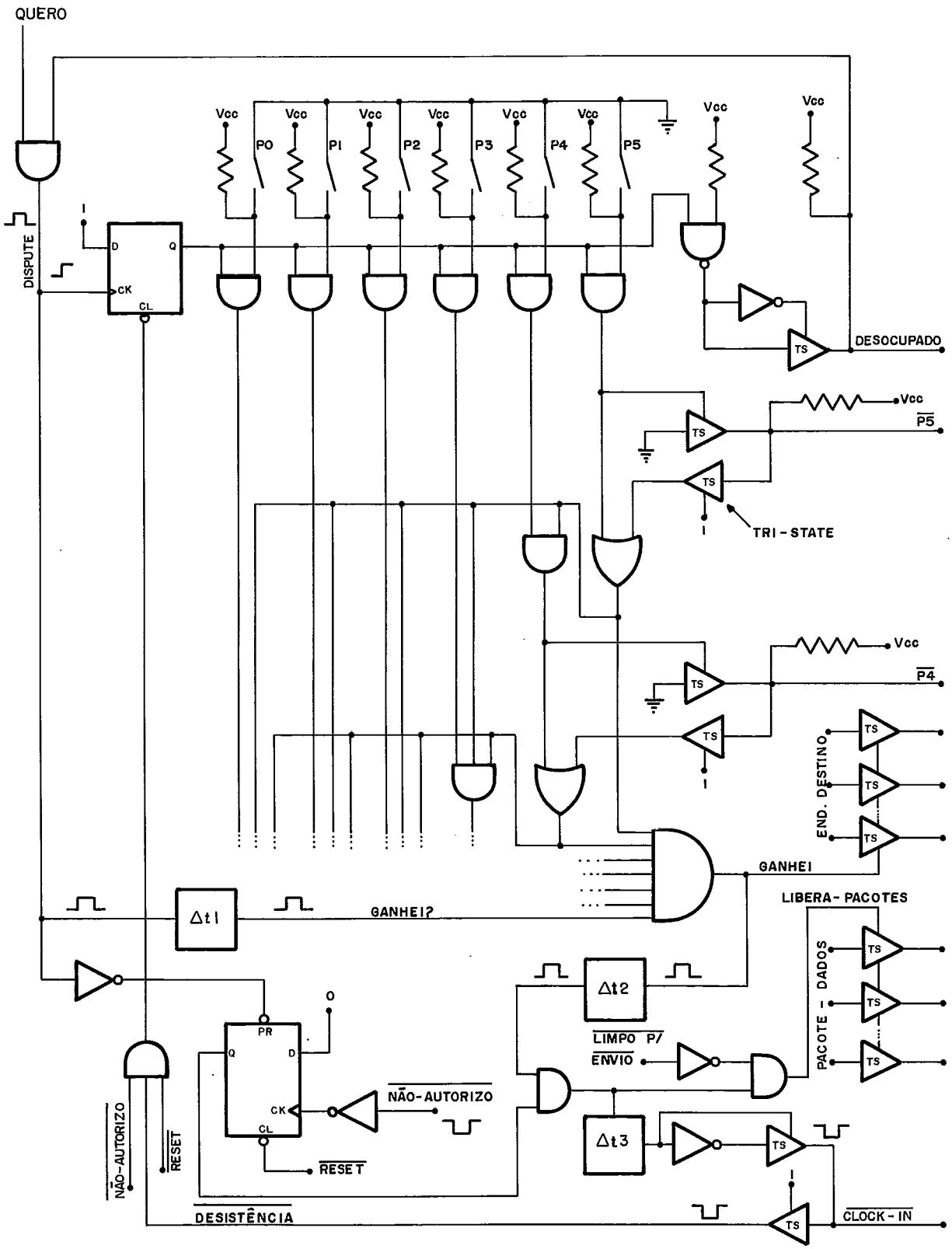


FIGURA VII.8 - CIRCUITO DE ACESSO AO BARRAMENTO

rar o sinal *quero* e continuar sua operação. O processador de comunicação destino, identificando seu endereço, emitirá o sinal *limpo-para-envio*=0 ou *não-autorizo*=0, se o "buffer" de recepção estiver livre ou ocupado respectivamente. A chegada do sinal *não-autorizo*=0 provocará a liberação do barramento, fazendo *desistência*=0, e informando ao processador de comunicação que seu pedido não foi atendido. Logo em seguida ou noutro momento, o pedido deverá ser resubmetido para disputa. A chegada do sinal *limpo-para-envio*=0 permitirá a emissão do sinal *clock-in*=0 que acionará o "buffer" de recepção destino para receber o pacote, e provocará a liberação do barramento para nova disputa.

No sentido de situar melhor o comportamento do circuito de acesso ao barramento, consta na figura (VII.10) o diagrama de tempo desse circuito para uma situação de acesso normal.

VII.2.3.3 - EXPANSÃO DO BARRAMENTO E INTERCONEXÃO DE REDES

O problema de expansão do barramento pode ser resolvido com a interconexão de redes através de um elemento denominado conector ("gateway"). A função do conector é receber pacotes de um barramento e colocá-los em outro com o devido endereçamento. Para isso, pode-se adotar diversas estruturas, sendo uma delas a estrutura em árvore, de acordo com a figura (VII.11).

O conector deverá entrar no lugar de um processador de comunicação e consumirá o endereço destino ao receber um pacote. Para poder continuar sua *viagem*, o pacote deverá ter embutido em si mesmo os passaportes necessários para que cada conector possa encaminhá-lo adequadamente. Os passaportes serão os endereços dos outros conectores intermediários e do processador de comunicação destino, e deverão desta forma reduzir a parte útil do pacote. O fato do pacote que circular entre barramentos precisar passar por conectores e ter uma informação útil menor faz com que a taxa de transmissão da informação sofra uma queda considerável.

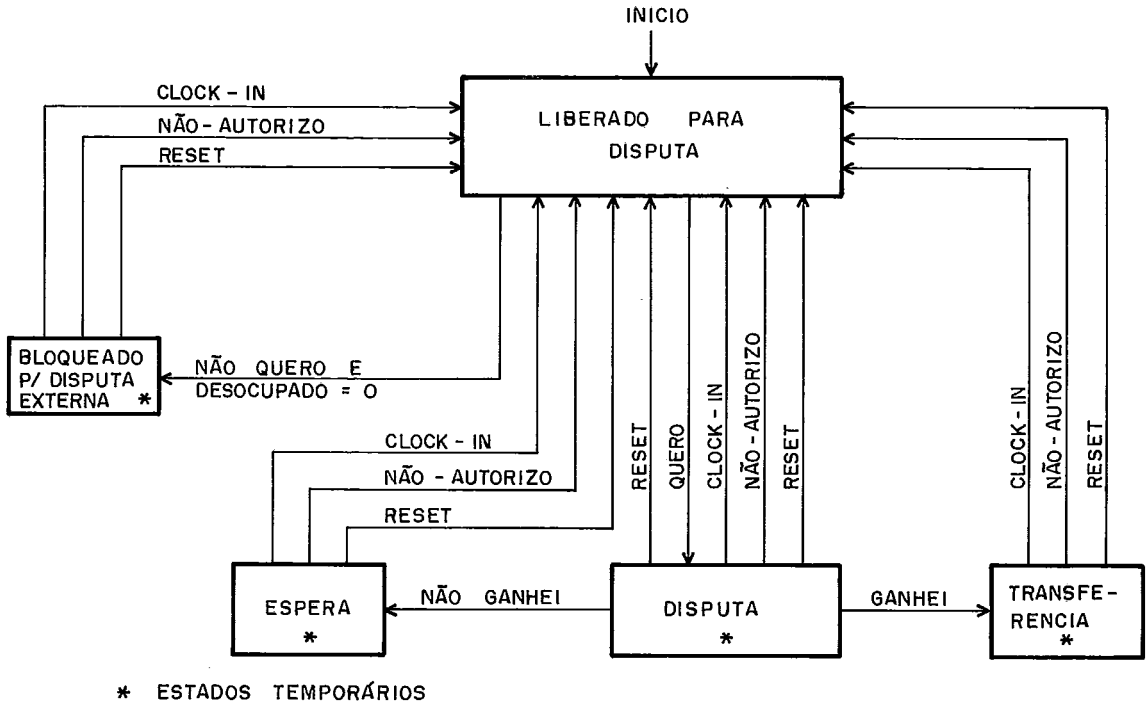


FIGURA VII . 9 - DIAGRAMA DE ESTADOS DO CIRCUITO DE ACESSO AO BARRAMENTO

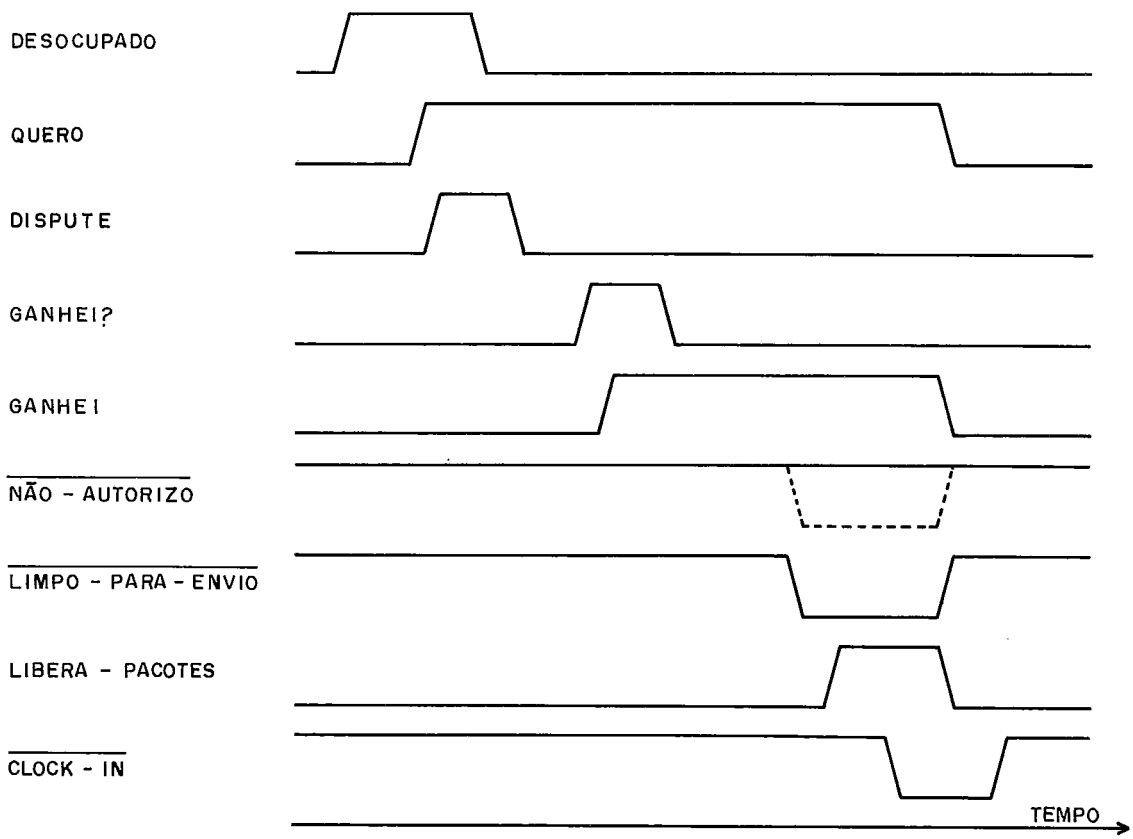


FIGURA VII . 10 - DIAGRAMA DE TEMPO DO CIRCUITO DE ACESSO AO BARRAMENTO

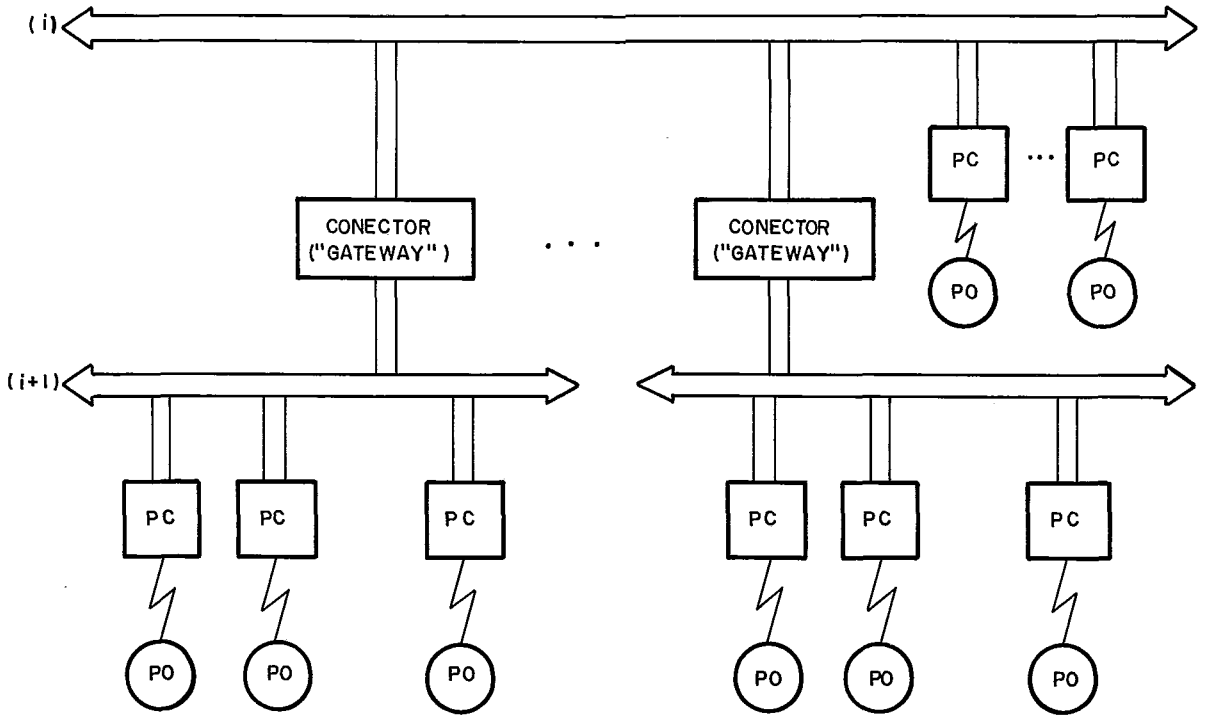


FIGURA VII.11 - UMA FORMA DE EXPANSÃO DO BARRAMENTO (INTERCONEXÃO DE REDES)

VII.2.4 - ASPECTOS DE SOFTWARE

O funcionamento do processador de comunicação baseia-se na existência de procedimentos que serão executados no microprocessador do módulo de controle das interfaces. Estes procedimentos é que colocarão o processador de comunicação nas condições iniciais, e depois, farão a monitoração dos sinais de controle e do estado das interfaces, cuidando da transferência interna das informações, e do acionamento de seus circuitos.

A ampliação dos procedimentos do processador de comunicação com primitivas de comunicação e sincronização entre processos, e com outras primitivas básicas de interesse de sistemas distribuídos implementa a parcela do núcleo referente ao processador de comunicação, ficando a parcela complementar para ser im

plementada no processador operador.

Como cada processador de comunicação reconhece seu próprio endereço individual, e um endereço comum a todos denominado endereço de difusão, a difusão será facilmente implementada, bastando que se faça a emissão de uma mensagem para o endereço destino de difusão.

C A P Í T U L O VIII

PROJETO DO SUPORTE BÁSICO DE UMA REDE EM LAÇO COM CONEXÃO PONTO A PONTO USANDO DETECÇÃO E RECUPERAÇÃO AUTOMÁTICA DE FALHAS

A rede em laço constitui-se numa das aplicações mais simples do subsistema de comunicação com conexão ponto a ponto, funcionando como bancada de trabalho. As decisões de roteamento, por exemplo, restringem-se à determinação do sentido do fluxo de cada informação; a existência de somente dois trajetos para alcançar-se o destino permite um tratamento do controle de fluxo com menor grau de complexidade do que seria se houvesse várias possibilidades do trajeto entre a origem e o destino. Tudo isto permite que o funcionamento da rede seja facilmente visualizado e controlado. Em contrapartida, a restrição do número de trajetos entre dois processadores faz com que a vulnerabilidade do sistema seja bastante alta em função da ocorrência de falhas no subsistema de comunicação.

Para a utilização da rede em laço no projeto de um sistema distribuído é necessário que o sistema possua mecanismos apropriados para o tratamento e recuperação automática de falhas, visando dotar o sistema com a mais alta confiabilidade possível. Além disso, em situações onde não se possa contornar falhas, o sistema deverá lançar mão de procedimentos de isolação das partes deterioradas, devendo reintegrá-las automaticamente tão logo desapareçam as causas do mau funcionamento.

Desta maneira, aproveitando-se a simplicidade da rede em laço, pode-se explorar com maior ênfase os aspectos de tolerância a falhas no suporte básico de um sistema distribuído, principalmente no que se refere ao software de comunicação localizado no núcleo.

VIII.1 - ESTRUTURA DA REDE

VIII.1.1 - HARDWARE

O hardware da rede em laço é formado pela conexão de vários processadores operadores ao subsistema de comunicação em laço com conexão ponto a ponto. Este subsistema de comunicação é composto por vários processadores de comunicação interligados entre si através de canais "full duplex", formando um laço.

O processador de comunicação é o mesmo descrito no capítulo VII, seção VII.1, apresentando como diferença somente o número de ligações com outros processadores de comunicação, que para esse caso é igual a dois.

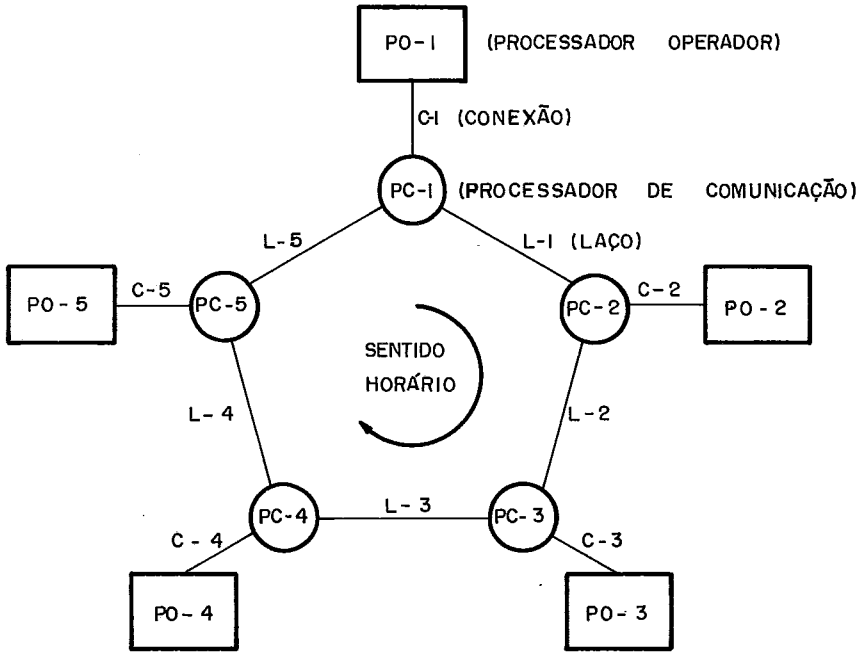
Para efeito de identificação e tratamento dos elementos da rede, tais como conexão com o processador operador, conexão com os outros processadores de comunicação, nó de comunicação, nó operador, etc., bem como para identificação das filas e "buffers" de cada processador de comunicação, definiu-se a convenção da figura (VIII.1) para a rede e para o processador de comunicação.

A conexão do processador de comunicação com o processador operador foi definida simplesmente por conexão, enquanto que as conexões do processador de comunicação com seus vizinhos idênticos foram denominadas laços. O sentido horário foi tomado como base para a determinação dos índices de identificação dos elementos da rede e do processador de comunicação.

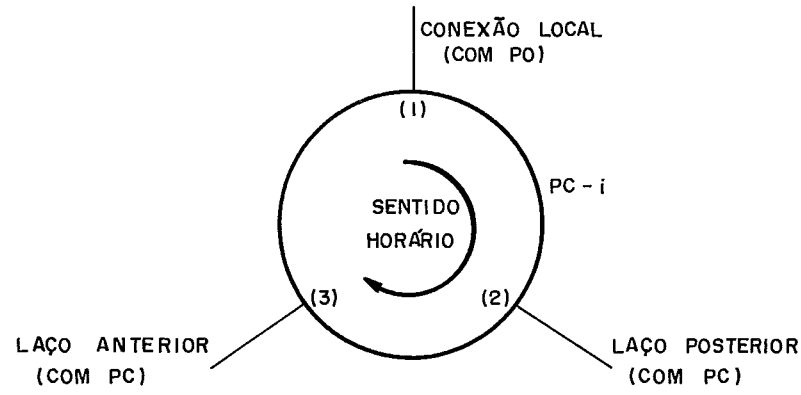
VIII.1.2 - A QUESTÃO DAS FALHAS

A rede em laço pode apresentar falhas de hardware ou de software localizadas em pontos diversos do sistema, dentre os quais: processador de comunicação, processador operador, laços e conexões. As falhas podem caracterizar-se como transitórias, quando provocadas por interferências momentâneas, ou como permanentes, quando forem de longa duração.

Para que a rede pudesse superar falhas transitórias e contornar falhas permanentes, optou-se por dotar o sistema com



a) ÍNDICES DOS ELEMENTOS DA REDE EM LAÇO COM 5 NÓS



b) ÍNDICES DOS ELEMENTOS DO PC ASSOCIADOS COM SUAS LIGAÇÕES EXTERNAS

FIGURA VIII.1 - MODELAMENTO ADOTADO PARA A REDE E PARA O PC

primitivas de comunicação que contivessem procedimentos de insistência e de mudança automática de rota. Os procedimentos de insistência, obtidos pela repetição da execução das operações, correspondem à redundância temporal. Além disso, visando manter o sistema funcionando, mesmo na presença de falhas permanentes que possam exigir manutenção corretiva, introduziu-se a isolação e a reintegração automáticas dos elementos envolvidos.

Ítens como saída por tempo ("time-out") e número máximo de repetições de operações, encontrados principalmente na elaboração das primitivas de comunicação, são utilizados para a superação de falhas transientes e para a identificação de falhas permanentes.

Cada processador de comunicação possui uma tabela indicativa da situação dos elementos da rede, cuja função é fornecer as informações necessárias ao estabelecimento da rota para que um pacote possa alcançar o destino. A existência de informações sobre pontos defeituosos da rede permitirá que o roteamento contorne tais pontos. Enquanto a tabela não estiver atualizada, qualquer falha no subsistema de comunicação deverá ser contornada por mudança automática de rota. De maneira análoga, a reintegração de qualquer elemento do subsistema de comunicação só estará efetivada completamente quando todas as tabelas estiverem atualizadas com a nova situação.

VIII.1.3 - ASPECTOS DE ROTEAMENTO

Na rede em laço, um processador de comunicação poderá receber pacotes de informação de seus vizinhos do subsistema de comunicação e do processador operador a ele ligado.

Um pacote proveniente de outro processador de comunicação poderá ser encaminhado para o processador de comunicação seguinte, respeitando-se o sentido da trajetória, ou ao processador operador local; esta decisão de encaminhamento do pacote dependerá exclusivamente do campo de destino do pacote que chegar: se o destino ainda não tiver sido alcançado, o pacote deverá continuar o seu trajeto pela rede em laço; se o destino for alcançado, o pacote deverá ser encaminhado ao processador operador local. De qualquer maneira, tais procedimentos não configuram de

cisão de roteamento por não envolverem a mensagem na sua origem.

Um pacote proveniente do processador operador origem poderá ser encaminhado ao destino através de dois trajetos diferentes na rede em laço: o trajeto no sentido horário ou o trajeto no sentido anti-horário. Esta decisão será tomada de acordo com a determinação do trajeto de menor distância, tendo como base tabelas de distâncias ou expressões de cálculo equivalentes, e levando em conta a presença de falhas. Assim, optou-se por utilizar roteamento com as seguintes características:

- fixo, com adaptações periódicas para casos de falhas permanentes;
- distribuído;
- usando o algorítmo determinístico do menor trajeto.

Para o cálculo do menor trajeto do nó origem ao nó destino, que não passasse por nenhum elemento com falha, utilizou-se a modelagem da figura (VIII.1) e as convenções e parâmetros definidos em seguida.

As convenções adotadas são as seguintes:

N = número de nós de comunicação da rede;

O = índice do processador origem;

D = índice do processador destino;

LF = índice do laço com falha;

H = sentido horário;

AH = sentido anti-horário;

MOD = resto da divisão inteira do antecedente pelo consequente.

Os parâmetros utilizados são os seguintes:

a) Distância do nó origem ao nó destino no sentido horário:

$$D_{(O-D)H} = (D + N - O) \text{ MOD } N;$$

b) Distância do nó origem ao nó destino no sentido anti-horário:

$$D_{(O-D)AH} = N - D_{(O-D)H};$$

c) Distância do nó origem a um laço com falha (incluindo ele próprio) no sentido horário:

$$D_{(O-LF)H} = ((LF + N - O) \text{ MOD } N + 1;$$

d) Distância do nó origem a um laço com falha (incluindo ele próprio) no sentido anti-horário:

$$D_{(O-LF)AH} = N - D_{(O-LF)H} + 1;$$

e) Nó posterior ao nó i:

$$NO\text{-}POST_i = (i \text{ MOD } N) + 1;$$

f) Nó anterior ao nó i:

$$NO\text{-}ANTI_i = (i - 2 + N) \text{ MOD } N + 1;$$

g) Laço posterior ao nó i:

$$LAÇO\text{-}POST_i = i;$$

h) Laço anterior ao nó i:

$$LAÇO\text{-}ANTI_i = (i - 2 + N) \text{ MOD } N + 1.$$

O algorítmo de roteamento, constante no apêndice C, consultará primeiramente a situação do processador operador destino e de sua conexão com a rede, usando uma tabela local. Se o

processador operador destino e sua conexão com a rede estiverem em condições normais, então passar-se-á à determinação da rota, caso contrário, uma informação, indicando a impossibilidade de estabelecimento da rota, será devolvida.

Na fase de determinação da rota há uma consulta sobre a existência de alguma falha nos laços do subsistema de comunicação, usando também uma tabela local. Se existir alguma falha nos laços, será descoberto, através da tabela de falhas nos laços, o primeiro laço com falha no sentido horário e o primeiro no sentido anti-horário. Em seguida, procurar-se-á o trajeto isento de falhas, ligando a origem ao destino, que poderá ser o trajeto no sentido horário ou anti-horário, ou poderá não existir. Caso não exista nenhuma falha nos laços, a distância da origem ao destino será calculada nos dois sentidos e a menor será tomada, indicando esse sentido como decisão de roteamento.

Cabe ressaltar que, quando uma falha existir, mas ainda não tiver sido atualizada, haverá a determinação da rota normalmente. Nesse caso, os pacotes encaminhados por aquela rota serão bloqueados na falha e passarão por um dos três tratamentos a seguir: terão sua rota alterada por meio de reversão de sentido; serão devolvidos à sua origem; ou, em último caso, serão descarregados.

VIII.1.4 - FORMATO DOS PACOTES

Há dois tipos de pacotes que circulam pela rede: os pacotes curtos de controle, e os pacotes longos de informação. Um pacote curto de controle é formado unicamente por um cabeçalho de doze campos, enquanto um pacote longo de informação contém, além desse cabeçalho, um campo adicional de informação. O campo de informação será responsável pela transferência da mensagem ou de parte dela entre dois nós operadores.

Os campos de um pacote, referentes ao cabeçalho são os seguintes:

1 - destino;
2 - origem;
3 - tipo do pacote (pacote curto/pacote longo);

4 - controle, indicando a função do pacote;
5 - operação;
6 - tamanho da mensagem;
7 - número do pacote/número da sequência;
8 - indicação de devolução;
9 - indicação de falha;
10 - conexão que falhou;
11 - primeiro laço que falhou num sentido;
12 - primeiro laço que falhou noutro sentido.

Os campos 1 - *destino*, e 2 - *origem* são utilizados para endereçar ou identificar os pontos da rede formados pelo par PC-PO. Ao chegar no ponto destino, o pacote poderá ser consumido no PC ou remetido ao processador operador, dependendo do seu conteúdo e do protocolo adotado, conforme será visto na seção seguinte.

O campo 3 - *tipo do pacote* indica diretamente se o pacote em questão é de controle ou de mensagem. Embora o campo seguinte também possa dar tal indicação indiretamente, optou-se por manter esta redundância, uma vez que ela oferece maior facilidade de tratamento com pouca sobrecarga.

O campo 4 - *controle* é utilizado para caracterizar a função do pacote, sendo de extrema importância para a implementação do protocolo de comunicação entre os pontos terminais. Ele indicará o uso do pacote para transportar:

1 - anúncio (resumo da mensagem);
2 - reconhecimento positivo de anúncio;
3 - reconhecimento negativo de anúncio;
4 - reconhecimento positivo de mensagem;
5 - reconhecimento negativo de mensagem;
6 - autorização de anúncio;
7 - autorização de mensagem;
8 - indicação de fila de anúncios vazia;
9 - "check sum" (os próximos campos conterão o "check sum");
10 - comandos de controle da rede para tratamento de falhas;
11 - mensagem.

Antes de especificar o significado dos próximos campos, é conveniente fazer um breve resumo do protocolo baseado em anúncio, cujo detalhamento consta na seção seguinte. Num sistema, funcionando com o modelo clientes/servidor, onde os vários clientes encaminham requisições de operação com seus respectivos dados a um único servidor, optou-se por manter os dados na origem e mandar somente um resumo da mensagem ao destino. Este resumo denominado anúncio incluirá a requisição da operação a ser desempenhada pelo servidor. Conforme já foi explicado no capítulo VI, seção 2.4, isto seria uma forma de resolver a questão de utilização de "buffers". Esses anúncios emitidos por várias origens permaneceriam numa fila situada no processador de comunicação destino, de onde seriam escolhidos para atendimento com base em requisições do processador operador destino (servidor). Essa escolha poderia ser realizada com o uso de diversas políticas que poderiam levar em conta inclusive o tipo de requisição embutida em cada anúncio. O atendimento consistiria em autorizar o emitente do anúncio escolhido a enviar os dados complementares para que o servidor o atendesse.

Continuando a descrição dos campos do cabeçalho de um pacote, pode-se dizer que o campo 5 - *operação* apresenta vários significados: quando associado a um anúncio (controle = 1), ele estará indicando a operação a ser realizada pelo processador operador destino que poderá ser o servidor; quando associado a um pacote de comando de controle (controle = 10), ele estará indicando uma operação de tratamento de falha a ser executada pelo processador de comunicação destino; quando associado a um pacote de "check sum" (controle = 9), ele estará levando uma parte do "check sum" para o processador operador destino. As operações a serem realizadas pelo processador operador destino serão definidas a nível de aplicação do sistema. As operações a serem realizadas pelo processador de comunicação destino abrangerão comandos de atualização de tabelas de falha, consistindo pelo menos de:

1 - anotar falha de laço;
2 - anotar falha de conexão PC-PO;
3 - retirar falha de laço;
4 - retirar falha de conexão PC-PO.

O campo 6 - *tamanho da mensagem* tem sua aplicação no pa

cote de anúncio, permitindo ao processador operador destino fazer a previsão e controle do "buffer" necessário para receber a mensagem por ocasião de seu atendimento. No caso de pacote de "check sum", este campo conterá outra parcela do mesmo.

O campo 7 - *número do pacote/número de sequência* é utilizado para numerar os pacotes de mensagem emitidos, permitindo a remontagem da mensagem independentemente da ordem de recebimento dos mesmos; quando usado como número de sequência, este campo serve para indicar a insistência de atendimento de um determinado anúncio, ou o abandono do anúncio corrente e atendimento de um novo. Como número de sequência, este campo também poderá ser usado para indicar anúncios diferentes emitidos pela mesma origem e encaminhados ao mesmo destino.

Cabe salientar que os campos descritos até agora consistem das informações básicas usadas para a transferência de mensagens. Os campos seguintes serão usados como infraestrutura apropriada para o tratamento de falhas.

O campo 8 - *indicação de devolução* terá anotado se o pacote esgotou as tentativas de alcançar o destino sem conseguí-lo, sendo portanto devolvido à origem.

O campo 9 - *indicação de falha* informa simplesmente se o pacote no seu trajeto encontrou alguma falha.

Os campos 10, 11, e 12 conterão a identificação dos elementos que falharam durante o trajeto do pacote, ou que estão sendo comunicados para a atualização das tabelas de falha.

Esses campos deverão ser dimensionados de acordo com a arquitetura da rede e com algumas características da aplicação.

VIII.1.5 - PROTOCOLO BASEADO EM ANÚNCIO

Conforme já foi considerado rapidamente na seção anterior, optou-se neste trabalho pela manutenção das mensagens em "buffers" situados nos locais de origem, até que ocorressem as condições favoráveis para uma transferência sem riscos de rejeição ou perda. Para isto, foi preciso utilizar-se o anúncio para avisar o destino de que haveria mensagem pendente para ele em condições de ser transferida. Ainda mais, foi preciso desenvolver-se um protocolo de comunicação entre operadores baseado em

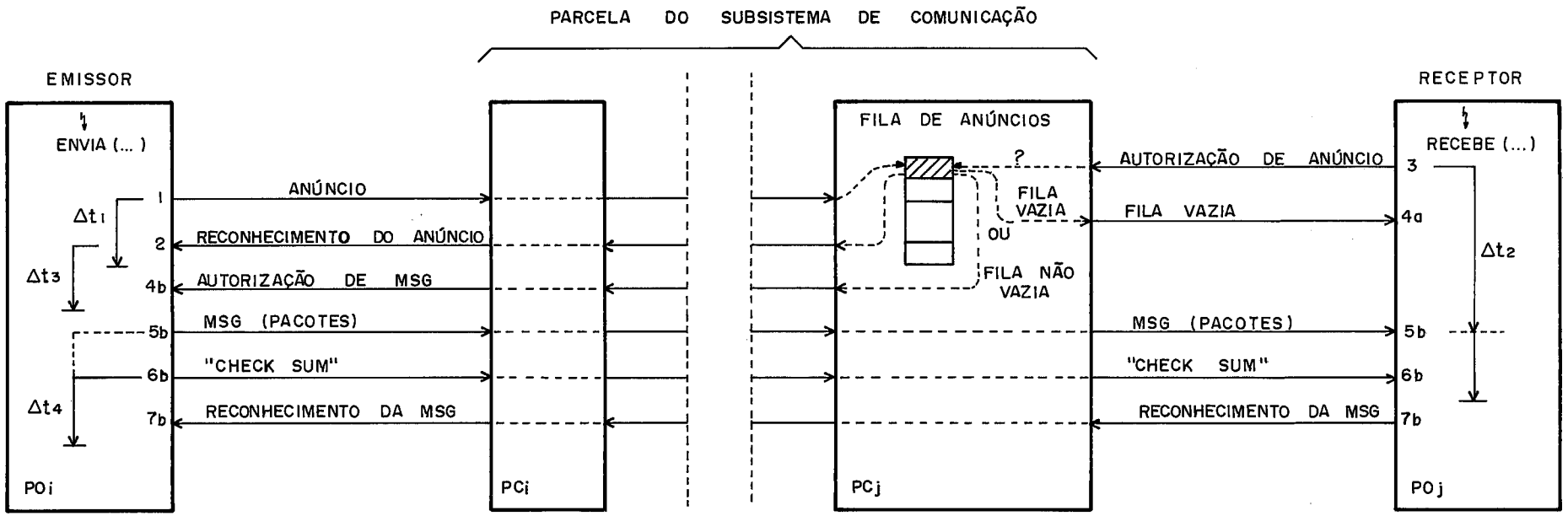
anúncio que assegurasse uma comunicação confiável. Uma primeira versão desse protocolo foi apresentada por KIRNER e SANTOS [86], usando os trabalhos de VON ISSENDORFF e GRÜNEWALD [192] como base.

Tendo em vista que o tipo de conexão PC-PO adotada no projeto prevê a interrupção do processador de comunicação pelo processador operador, mas não o contrário, resolveu-se localizar a fila de anúncios no processador de comunicação destino, satisfazendo adequadamente as necessidades da rede e do processador operador destino. Desta maneira, sempre que um processador operador origem quiser transmitir uma mensagem para um processador operador destino, ele deverá armazenar a mensagem num "buffer" local, preparar um anúncio, remetê-lo à fila localizada no processador de comunicação destino, e esperar pelo atendimento de anúncio por iniciativa do processador operador destino.

Para a visualização do protocolo, tomou-se uma comunicação entre dois processos situados em processadores diferentes, onde um corresponde a um cliente (processo emissor) e outro corresponde ao servidor (processo receptor). O processo emissor executará uma primitiva *envia* que, se for bloqueante, bloqueará o processo emissor até o processo receptor receber a mensagem, ou se for não bloqueante, deixará o processo emissor continuar tão logo o núcleo receba a mensagem a ser enviada. O processo receptor, por sua vez, executará uma primitiva *recebe* que, se for bloqueante, bloqueará o processo receptor até o recebimento da mensagem, havendo ou não mensagem pendente, ou se for não bloqueante, deixará o processo continuar tão logo tome conhecimento de inexistência de mensagem pendente. Haverá mensagem pendente sempre que a fila de anúncios do receptor não estiver vazia.

O protocolo utilizado entre processadores operadores, envolvendo os processadores de comunicação como elementos intermediários e sem levar em conta as várias possibilidades de falha, pode ser visto na figura (VIII.2). Cabe citar que, entre os processadores internos envolvidos na comunicação, existe um protocolo de nível mais baixo não considerado nessa análise. A implicação das falhas nesse protocolo será analisada e discutida posteriormente na seção referente ao desenvolvimento das primitivas de comunicação e sincronização.

O funcionamento detalhado do protocolo é descrito em



LEGENDA

MSG = MENSAGEM
PC = PROCESSADOR DE COMUNICAÇÃO
PO = PROCESSADOR OPERADOR
Δt = PERÍODO DE SAÍDA POR TEMPO

FIGURA VIII.2 - PROTOCOLO DE COMUNICAÇÃO ENTRE DOIS PROCESSADORES OPERADORES (USANDO ANÚNCIO)

seguida.

O processo emissor localizado no processador operador, contendo uma mensagem já preparada, fará uma chamada de núcleo, requisitando a execução da primitiva *envia*. A parcela do núcleo situada no processador operador, dependendo do tipo da primitiva e da implementação, poderá copiar a mensagem num "buffer" apropriado ou simplesmente assumir o controle da área referente à mensagem. Em seguida, essa parcela do núcleo, considerando os parâmetros da chamada e a própria mensagem, fará a montagem do anúncio e o remeterá num pacote curto de controle para o processador de comunicação existente no local. Ali, o pacote receberá o tratamento complementar por parte da outra parcela do núcleo, que neste caso consistirá do encaminhamento adequado do anúncio, visando alcançar o destino. Ao chegar ao processador de comunicação destino, o anúncio será colocado na fila de anúncios ali existente, e um pacote de reconhecimento positivo de anúncio será remetido de volta. Se, por acaso numa situação imprevista, o anúncio não puder ser colocado na fila, então será remetido um reconhecimento negativo de anúncio. Se receber um pacote de reconhecimento positivo, o emissor ficará aguardando a chegada de autorização de mensagem, mas se receber um pacote de reconhecimento negativo, o emissor poderá tomar uma dentre várias providências como desistir da emissão, retransmitir o anúncio, esperar um tempo e retransmitir o anúncio, etc., dependendo do tipo da primitiva e da implementação.

Por outro lado, o processo receptor, em qualquer instante e independentemente do emissor, poderá fazer uma chamada de núcleo, requisitando a execução da primtiva *recebe*. A parcela do núcleo situada no processador operador remeterá então um pacote de autorização de anúncio para o processador de comunicação com o qual está ligado. Aí, a parcela do núcleo, referente ao processador de comunicação, consultará a fila de anúncios alí localizada, verificando se existe ou não anúncios. Se a fila estiver vazia, um pacote de controle, indicando fila vazia, será remetido de volta ao processador operador, caso contrário, o anúncio será transformado num pacote de autorização de mensagem, e transferido a outro processador de comunicação de forma a seguir caminho para alcançar o processador operador destino.

Ao receber uma autorização de mensagem, o emissor que

estava em estado de espera fará a decomposição da mensagem em diversos pacotes e os remeterá ao emissor. Durante esta tarefa, o "check sum" será calculado, e remetido por último, finalizando a transferência da mensagem. O receptor, recebendo os pacotes, remontará a mensagem, calculando também o "check sum" para poder compará-lo com o que vier do emissor. Se a comparação de "check sum" resultar em igualdade, então o receptor enviará ao emissor um pacote de reconhecimento positivo da mensagem, encerrando esta comunicação, caso contrário, haverá a emissão de reconhecimento negativo de mensagem que, dependendo do tipo de primitiva e da implementação, poderá resultar em retransmissão da mensagem, término da comunicação, etc.

A figura (VIII.2) também contém uma indicação de saída por tempo ("time-out") que, se acontecer, deverá ser seguida de ações que terminem, recuperem, ou alterem, de alguma maneira, o protocolo em andamento. A saída por tempo ficará latente sempre que a contagem de tempo a partir de um evento for disparada. A saída por tempo ocorrerá quando o tempo esgotar-se, ou será neutralizada quando um evento aceitável aparecer antes dela. O tratamento detalhado da saída por tempo será feito na seção relativa ao desenvolvimento das primitivas de comunicação e sincronização.

A maior parte das ações relacionadas com o protocolo são originadas e tratadas pela parcela do núcleo localizada nos processadores operadores. Caberá à parcela do núcleo, situada nos processadores de comunicação, cuidar na maior parte das vezes da transferência de pacotes de forma a permitir que fluam da origem para o destino. No entanto, percebe-se diretamente pelo protocolo que o anúncio e a autorização de anúncios, embora originados nos processadores operadores, são tratados a nível de processador de comunicação. O tratamento das falhas detectadas no subsistema de comunicação também constitui-se em atividade realizada a nível de processador de comunicação, exercendo em certos casos alguma influência a nível de processador operador como, por exemplo, a atualização de tabelas e o término forçado de primitivas de comunicação. A ocorrência de falhas no subsistema de comunicação, em momentos e locais que impeçam o prosseguimento da transferência de pacotes, fará com que esses pacotes sejam devolvidos ao processador que os tiver emitido. No caso de devo

lução de pacotes de reconhecimento positivo de anúncio e de autorização de mensagem, o processador de comunicação que tiver emitido esses pacotes deverá localizar o anúncio correspondente na fila de anúncios e destruí-lo. Isto deverá ocorrer para evitar que tais anúncios fiquem ocupando espaço na fila, e mesmo para não permitir que sejam atendidos sem necessidade; quando as primitivas que encaminharem os anúncios não receberem o respectivo reconhecimento ou autorização de mensagem, eles serão tomados como perdidos, sendo abandonados ou retransmitidos em novas tentativas. Qualquer pacote, que deva ser tratado pelo processador de comunicação de forma diferente da transferência pura e simples, será considerado como comando para o processador de comunicação.

VIII.1.6 - ASPECTOS DO NÚCLEO

O núcleo do sistema, composto por suas diversas parcelas situadas no subsistema de comunicação e complementadas pelas parcelas correspondentes localizadas nos processadores operadores, caracteriza-se por oferecer a infraestrutura de software necessária ao desenvolvimento de sistemas operacionais distribuídos e de programas utilitários da rede. Essa infraestrutura de software coloca primitivas já implementadas à disposição dos processos do sistema localizados nos processadores operadores. Tais primitivas permitem, por exemplo, que a sincronização e comunicação entre processos ocorra sem o conhecimento dos detalhes da rede e da implementação dessas primitivas. Além disso, o núcleo apresenta outras primitivas de interesse do sistema, atuando em níveis próximos do hardware, como é o caso de primitivas de entrada e saída, primitivas de configuração do sistema, etc. A execução de uma primitiva poderá ser disparada por chamadas provenientes do software ou do hardware. As chamadas por software serão feitas pelos níveis superiores do sistema como se fossem chamadas de procedimentos, e as chamadas por hardware ocorrerão através de interrupção.

O subsistema de comunicação com seu respectivo software funcionará como elo de ligação entre as parcelas do núcleo espalhadas pelo sistema, fazendo com que o núcleo apresente um com

portamento integrado. Além disso, o núcleo fará com que o sub sistema de comunicação seja transparente aos níveis superiores do sistema.

Num ambiente de rede em laço formado por uma estação de desenvolvimento, contendo periféricos e o sistema armazenado, e várias estações de trabalho com configuração mínima, há necessi dade de um conjunto mínimo de primitivas relacionadas com a re de. Esse conjunto consiste de primitivas de configuração e parti da do sistema e primitivas de comunicação e sincronização.

As primitivas de configuração e partida do sistema são as seguintes:

- envia programa para carregamento remoto;
- recebe programa para carregamento;
- envia sinal de início de execução;
- recebe sinal de início de execução;
- envia requisição de reconfiguração;
- recebe requisição de reconfiguração.

As primitivas de comunicação e sincronização entre pro cessos são as seguintes:

- envia mensagem;
- recebe mensagem.

As primitivas de configuração e partida do sistema ser vem para que a estação de desenvolvimento distribua os programas entre as estações de trabalho e comande o início de sua execu ção. Admitindo-se que a energização das estações coloque cada estação de trabalho em estado de espera através da chamada da primitiva *recebe programa para carregamento*, a estação de desen volvimento poderá acessar os vários programas a serem distribuí dos e executar para cada um a primitiva *envia programa para car regamento*. Isto fará com que cada programa seja transferido para a estação de trabalho destino para ser carregado nas áreas de me mória apropriadas. Cada estação de trabalho, recebendo informa ções de controle do carregamento e o respectivo programa, fará a sua colocação na memória e em seguida entrará novamente em esta do de espera, devido à chamada da primitiva *recebe sinal de iní cio de execução*. Após executar todos os carregamentos remotos,

a estação de desenvolvimento disparará o início de execução do sistema através da primitiva *envia sinal de início de execução*, que, se for implementada como primitiva de difusão, desencadeará todas as partidas, senão terá que ser repetida para cada estação de trabalho.

As estações de trabalho, recebendo o disparo, iniciarão a execução de seus programas que poderão comunicar-se através das primitivas de comunicação e sincronização. As primitivas de reconfiguração, quando implementadas, permitem a alteração total ou parcial do sistema, exigindo no entanto muito cuidado no tratamento do sistema, devido à possibilidade de aparecimento de órfãos. Prevê-se que, ao receber uma requisição de reconfiguração, uma estação de trabalho abandone seu programa e caia num estado de novo carregamento, executando as primitivas *recebe programa para carregamento* e *recebe sinal de início de execução*. Para que uma estação de trabalho da rede em laço em questão possa aceitar requisições de reconfiguração, o processador operador correspondente deverá executar periodicamente a primitiva *recebe requisição de reconfiguração* que, obrigatoriamente, deverá ser não bloqueante. Isto poderá ser feito através de interrupções periódicas provocadas por um relógio, ou pela colocação sistemática da chamada desta primitiva ao longo do programa. Assim, se não houver requisição de reconfiguração pendente, a execução da primitiva não terá nenhum efeito, deixando que o programa continue normalmente. O funcionamento dessas primitivas dependerá da existência de estrutura de dados apropriadas, e de ações tomadas pelas parcelas do núcleo situadas no processador operador e no processador de comunicação.

Algumas primitivas de comunicação e sincronização utilizadas para a interação entre os processos do sistema foram implementadas de maneira a satisfazer o protocolo baseado em anúncio. Para isto, o programa supervisor do processador de comunicação teve que incorporar uma fila de anúncios e estruturas de controle adicionais, e assumir o comportamento exigido pelo protocolo. Por outro lado, a parcela do núcleo do processador operador consistiu do agrupamento das primitivas que podem ser chamadas pelos processos. As implementações do supervisor do processador de comunicação, referente a uma parcela do núcleo, e do conjunto das primitivas de comunicação e sincronização locali

zado no processador operador, referente a outra parcela do núcleo, serão consideradas nas seções seguintes.

Cabe salientar que nessa implementação foi considerado que cada processador operador executará um único processo, o qual, ao ficar bloqueado, poderá permanecer em espera ocupada; por isso, a chamada da parte do núcleo existente no processador operador, quando realizada pelos processos, será feita como chamada de procedimento sem necessidade de salvar o contexto. No caso de existir chamada de núcleo através de interrupção, o processo corrente deverá ser salvo e suspenso para poder ser restaurado e continuar sua execução depois do tratamento da chamada. Numa situação genérica de vários processos atuarem em uma mesma estação em regime de multiprogramação, o salvamento deverá ser feito para qualquer chamada de núcleo, cujo término será sempre precedido da escalação de novo processo que será restaurado para continuação.

Se não tiver sido bloqueado, o processo suspenso por ocasião da chamada de núcleo poderá eventualmente ser escalado em seguida.

VIII.2 - ESTRUTURA DO SUPERVISOR DO PROCESSADOR DE COMUNICAÇÃO

O supervisor do processador de comunicação, correspondendo à parcela do núcleo do sistema relacionada com o subsistema de comunicação, contém a implementação de parte da infraestrutura necessária ao funcionamento das primitivas do sistema. Conforme já foi mencionado na seção 1.2 do capítulo VII, o programa supervisor é o responsável pelo controle de fluxo de informação no subsistema de comunicação, cuidando do estabelecimento de rotas para os pacotes de mensagens, da manipulação das filas, e da recepção e transmissão dos pacotes.

Para poder executar sua tarefa na rede em laço com conexão ponto a ponto, o supervisor deverá contar com uma estrutura de dados de comunicação, repetida para cada ligação do processador de comunicação com o processador operador ou com seus vizinhos, e com uma única estrutura de dados de uso comum.

A estrutura de dados de comunicação para uma ligação do

processador de comunicação com um elemento externo consiste do seguinte:

- "buffer" de recepção;
- variável indicando se o "buffer" de recepção está cheio;
- fila de pacotes curtos de controle;
- fila de pacotes longos de informação.

A estrutura de dados de uso comum contém os seguintes elementos:

- "buffer" de trabalho;
- fila de anúncios;
- anúncio que está sendo atendido;
- variável contendo o número de sequência atual;
- tabela indicativa de falhas nos laços e nas conexões com processadores operadores;
- variável indicando se existe alguma anotação de falha na tabela;
- marca de varredura da transmissão;
- ponteiro de varredura da transmissão.

O tratamento da estrutura de dados de comunicação é feito com o uso de índices, cuja correspondência com as ligações é a seguinte: índice igual a 1 - ligação com o processador operador; índice igual a 2 - ligação com o processador de comunicação posterior no sentido horário; e índice igual a 3 - ligação com o processador de comunicação anterior no sentido anti-horário.

Embora a atividade predominante do supervisor seja manipular os pacotes de forma a fazê-los chegar ao seu destino, alguns pacotes são caracterizados como comandos ao processador de comunicação, exigindo o tratamento adequado pelo supervisor. Esses comandos são os seguintes:

- chegada de um anúncio ao processador de comunicação destino, vindo do subsistema de comunicação;
- chegada de uma autorização de anúncio ao processador de comunicação, vinda do processador operador associado;

- chegada de pacote devolvido por causa de falha, referente a reconhecimento positivo de anúncio ou a autorização de mensagem;
- chegada de comandos de controle da rede, relativos a atualização da tabela de falhas.

O programa supervisor, conforme algorítmo do apêndice D, deverá funcionar de forma análoga àquela descrita na seção 1.2 do capítulo VII, complementado com o tratamento das falhas e dos comandos para o processador de comunicação. Assim, o supervisor deverá passar alternadamente pelas etapas de recepção e transmissão.

Na etapa de recepção, o supervisor fará a varredura de todos os "buffers", verificando se a recepção já está terminada. Para cada "buffer" com recepção terminada, será feito sua transferência para o "buffer" de trabalho e o seu teste para determinar se o pacote será transferido para uma fila de transmissão, ou se o mesmo corresponde a um comando para o processador de comunicação. Se for o caso de transferência pura e simples, passar-se-á por uma fase de decisão que determinará aonde colocar o pacote. Quando o pacote for proveniente do processador operador, a transferência deverá usar o algorítmo de rota para a determinação do sentido de encaminhamento do pacote. Se for o caso de comando para o processador de comunicação, o supervisor deverá tomar as providências cabíveis, incluindo a emissão de respostas se for preciso.

Em seguida, depois de esgotada a fase de recepção, deverá acontecer a etapa de transmissão. A partir da marca de varredura da transmissão, o supervisor procurará o próximo par de filas, onde existir alguma fila não vazia, e comandará a transmissão de um pacote dessa fila pela linha serial. Se a transmissão não puder efetivar-se por causa de alguma falha, o supervisor inspecionará o pacote para ver se o mesmo já teve sua trajetória revertida por algum insucesso anterior; se ainda não houve nenhuma reversão, então a falha será anotada no pacote, e o mesmo passará a percorrer um trajeto de sentido revertido ao ser colocado na fila de "buffers" de transmissão oposta; se já tiver havido reversão, o supervisor anotará esta outra falha e transformará o pacote em devolução, encaminhando-o para a origem. Sem

pre que detectar uma falha na fase de transmissão, o supervisor inspecionará sua tabela de falhas, procurando pela anotação da falha; se a mesma não estiver anotada, então ocorrerá a anotação e os outros supervisores serão avisados da presença dessa falha; se a falha já estiver anotada, o aviso será remetido somente para o supervisor situado na estação que tiver originado o pacote, cuja transmissão não tenha sido possível. Cabe ressaltar que a escolha do pacote para transmissão deverá, sempre que possível, recair na fila de pacotes curtos de controle. Depois de feita a transmissão, reversão, ou devolução, a marca de varredura da transmissão será atualizada, indicando a fila manipulada. Depois disso, passar-se-á novamente para a etapa de recepção. Se, por acaso, a busca de fila de transmissão não vazia não tiver sucesso ao longo de um ciclo de varredura, passar-se-á também para a etapa de recepção.

A cada período de tempo ou número de ciclos do supervisor, a ser definido pelo projetista, o supervisor eliminará as anotações de falhas constantes nas suas tabelas para que haja oportunidade do subsistema de comunicação de testar os seus laços e conexões. Com isto, haverá tentativas de passagem pelas ligações que anteriormente apresentavam falhas. Se a falha estiver persistindo, haverá um aviso aos outros elementos da rede ou a um elemento específico, dependendo da situação da anotação da falha no processador de comunicação que detectar a falha. Se a falha tiver desaparecido, então a eliminação da anotação terá sido bem sucedida. Cada supervisor, atuando desta maneira, manterá as tabelas de falhas atualizadas, fazendo com que o algorítmo de rota apresente bom funcionamento.

A estruturação do programa supervisor em níveis de implementação encontra-se no apêndice E, consistindo da relação de procedimentos de nível mais baixo que cada procedimento pode chamar diretamente.

VIII.3 - DESENVOLVIMENTO DAS PRIMITIVAS DE COMUNICAÇÃO E SINCRONIZAÇÃO

A implementação das primitivas de comunicação e sincro

nização deve dar-se de acordo com o protocolo baseado em anúncios. Para isto, cada primitiva contém uma parcela de código no processador operador e outra parcela inserida no supervisor do processador de comunicação, ambas fazendo parte do núcleo. Enquanto a parcela da primitiva correspondente ao processador operador é formada por várias operações que devem ser disparadas em determinadas sequências, a parte da primitiva, relativa ao processador de comunicação, restringir-se-á ao tratamento dos comandos para o processador de comunicação feito no supervisor. A sequencialização das operações que compõem a primitiva a nível de processador operador será feita por um trecho de programa denominado controlador de sequência das primitivas.

Neste trabalho, optou-se pela implementação das primitivas *envia bloqueante*, *recebe não bloqueante* e *recebe bloqueante*. A primitiva *envia não bloqueante*, embora muito importante para a implementação de mecanismos de difusão, foi considerada e discutida a nível de especificação por exigir uma ampliação no dimensionamento dos "buffers" e nas atividades do supervisor.

Com relação ao endereçamento, a fila de anúncios e o protocolo adotado previram a utilização de endereçamento explícito direto, simétrico, e assimétrico. O fato do anúncio permanecer numa fila próximo ao receptor faz com que a escolha de um anúncio para atendimento recaia sobre qualquer um, ou sobre um anúncio de uma origem especificada, configurando assim a assimetria ou simetria do endereçamento.

Com relação ao tratamento de falhas, as primitivas foram especificadas em duas classes: primitivas com detecção de falhas; e primitivas com recuperação automática de falhas. O projeto do supervisor do processador de comunicação foi desenvolvido para suportar as duas classes de primitivas.

VIII.3.1 - CONTROLADOR DE SEQUÊNCIA DAS PRIMITIVAS

Cada parcela de primitiva situada no processador operador e estruturada com base num diagrama de estados consiste de um conjunto de operações de transição e de um controlador de sequência da primitiva. Esse controlador de sequência é quem determina os pontos de espera da primitiva, bem como a ordem de

execução das operações de transição.

Genericamente, o controlador de sequência de uma primitiva deverá funcionar de acordo com o algorítmo (VIII.1).

Toda vez que uma primitiva for chamada por um processo, esta chamada será transformada num estado inicial de entrada no controlador de sequência daquela primitiva. O controlador, dependendo dos eventos de entrada, irá executando as operações de transição, alterando os estados, e permanecendo à espera de novos eventos, até atingir o estado final, quando retornará o controle para o processo que tiver chamado a primitiva.

VIII.3.2 - PRIMITIVAS COM DETECÇÃO DE FALHAS

As primitivas com detecção de falhas caracterizam-se por terminar sua execução, sempre que for detectado um funcionamento anormal decorrente de falhas transitórias ou permanentes. O funcionamento anormal será detectado através de: conhecimento explícito da falha anotado em devolução de pacotes; ações previstas no protocolo como reconhecimento negativo; e de saída por tempo ("time-out"). A devolução de pacotes com a anotação das falhas ocorrerá sempre que um pacote não conseguir atingir o destino, devido à existência de falhas nos trajetos possíveis entre a origem e o destino.

Os tempos envolvidos nas saídas por tempo poderão ser definidos pelo usuário ou ajustados ao nível do núcleo. O sistema poderá usar tempos padrão para operações que atuem somente a nível de subsistema de comunicação, devido à possibilidade de previsão de sua duração máxima. Ficará para o usuário a determinação de tempos que envolvam operações de processador operador dependentes do comportamento do sistema.

Após o término normal ou forçado de uma primitiva, uma variável de "status" estará indicando o motivo da saída.

VIII.3.2.1 - PRIMITIVA *ENVIA BLOQUEANTE* COM DETECÇÃO DE FALHAS

No processador operador, a primitiva *envia bloqueante* com detecção de falhas deverá funcionar de acordo com o diagrama de estados da figura (VIII.3).

```
estado ← estado inicial;
condição de saída ← 0;
repita
   execute o código do estado igual a
      estado inicial: início
                      :
                      estado ← novo estado
                      fim
      ⋮
      outros estados: início
                      esperar a ocorrência de um evento aceitável;
                      analisar o evento;
                      executar a operação de transição;
                      estado ← novo estado
                      fim
      ⋮
      estado final: início
                    :
                    condição de saída ← 1
                    fim
      fim
até condição de saída = 1
```

ALGORÍTMO VIII.1 - ALGORÍTMO GENÉRICO DO CONTROLADOR DE SEQUÊNCIA DA PRIMITIVA

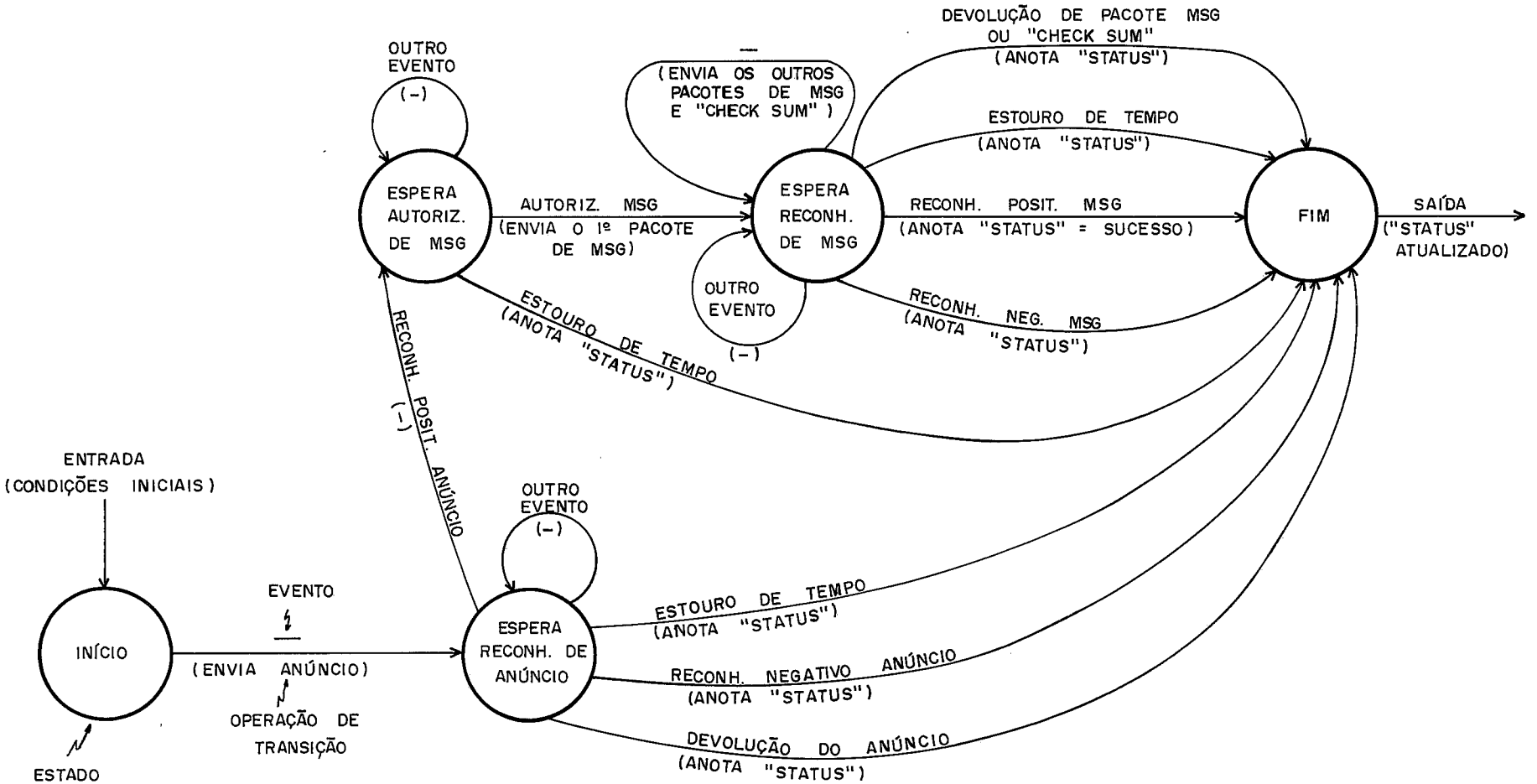


FIGURA VIII . 3 - DIAGRAMA DE ESTADOS DA PRIMITIVA "ENVIA BLOQUEANTE" COM DETECÇÃO DE FALHAS

Neste caso, as várias possibilidades que a variável de "status" poderá assumir são as seguintes:

- estouro de tempo da espera pelo reconhecimento de anúncio;
- reconhecimento negativo de anúncio;
- estouro de tempo da espera pela autorização de mensagem;
- reconhecimento negativo de mensagem;
- estouro de tempo da espera pelo reconhecimento de mensagem;
- falha explícita na rede (devolução);
- sucesso (recebimento do reconhecimento positivo de mensagem).

Estando em qualquer estado, a ocorrência de um evento previsto provocará a execução da operação de transição e a mudança de estado. Quando o evento for nulo, a execução da operação de transição e a própria transição serão automáticas.

A primitiva será bloqueante enquanto as condições normais de funcionamento persistirem, mas quando ocorrer uma falha, um reconhecimento negativo de anúncio ou mensagem, ou um estouro de tempo, a primitiva será encerrada.

VIII.3.2.2 - PRIMITIVA *RECEBE NÃO BLOQUEANTE* COM DETECÇÃO DE FALHAS

No processador operador, a primitiva *recebe não bloqueante* deverá funcionar de acordo com o diagrama de estados da figura (VIII.4).

As várias possibilidades que a variável de "status" poderá assumir são as seguintes:

- estouro de tempo da espera pela resposta;
- fila vazia;
- emissão de reconhecimento negativo de mensagem;

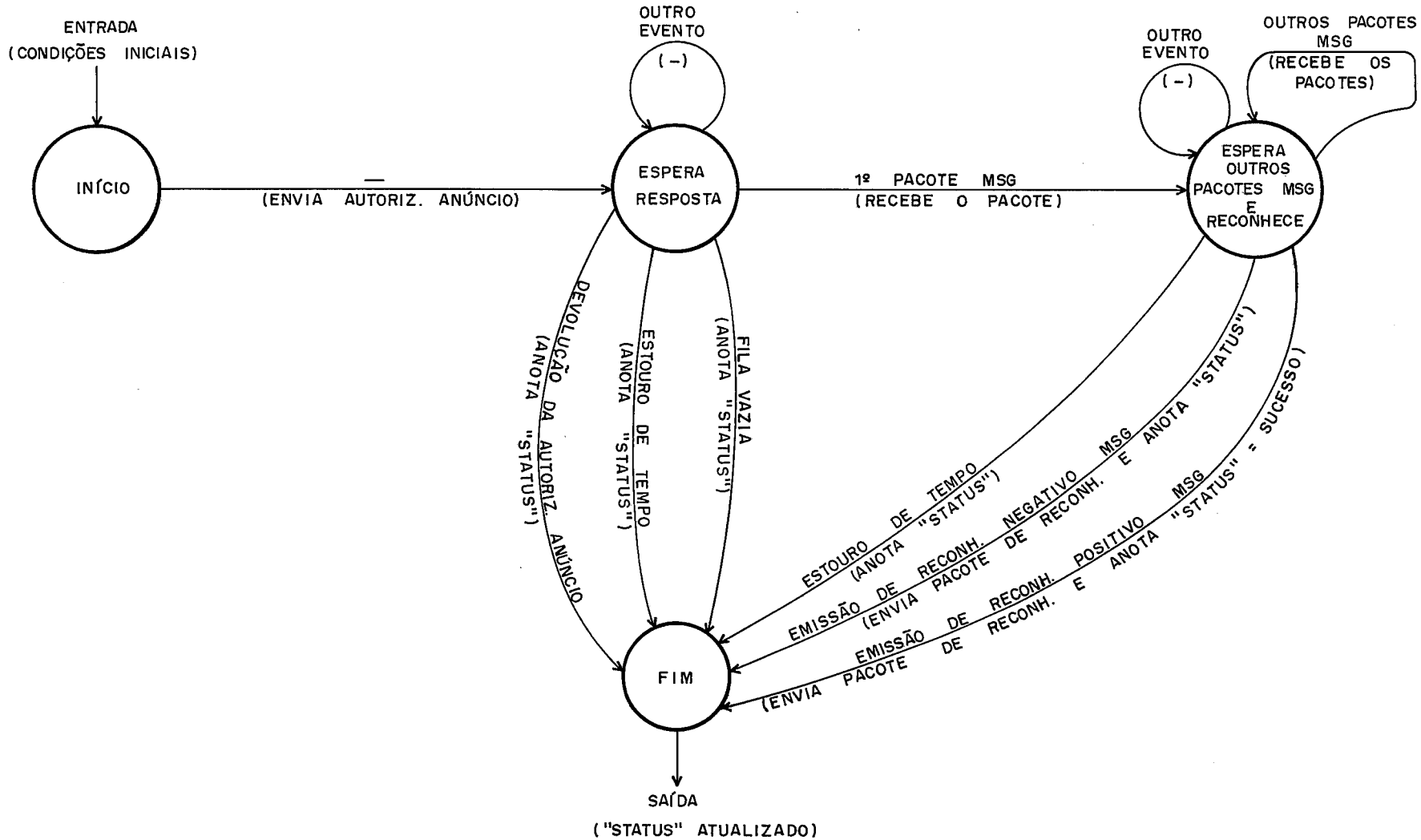


FIGURA VIII . 4 - DIAGRAMA DE ESTADOS DA PRIMITIVA "RECEBE NÃO BLOQUEANTE" COM DETECÇÃO DE FALHAS.

- estouro de tempo da espera pelos pacotes de mensagem e de "check sum";
- falha explícita na rede (devolução);
- sucesso (emissão de reconhecimento positivo de mensagem).

Num funcionamento normal, as duas únicas saídas possíveis serão aquelas, cuja variável de "status" contenha sucesso ou fila vazia. No caso de sucesso, uma mensagem terá sido recebida e o processo receptor poderá continuar sua execução. No caso de fila vazia, não existirá nenhum anúncio na fila e por conseguinte nenhum receptor querendo mandar mensagem para o receptor, de forma que o receptor continuará sua execução assim mesmo. Uma nova tentativa de recepção deverá ser de responsabilidade do receptor que, ao testar a variável de "status" configurando insucesso, poderá desviar a execução para repetir a primitiva.

VIII.3.2.3 - PRIMITIVA *RECEBE BLOQUEANTE* COM DETECÇÃO DE FALHAS

A implementação da primitiva *recebe bloqueante* com detecção de falhas poderá ser realizada dentro do núcleo com o uso da primitiva *recebe não bloqueante* inserida num ciclo de repetição da forma do algorítmo (VIII.2).

Para evitar que a repetição da primitiva *recebe não bloqueante*, utilizando autorização de anúncio, implicasse num esvaziamento de filas de anúncio, no caso de saídas anormais, optou-se pelo uso do número de sequência associado à operação de transição *envia autorização de anúncio*. Assim, a repetição da primitiva com mesmo número de sequência permitirá a insistência de atendimento de um anúncio, enquanto que, com número de sequência diferentes, haverá tentativas de atendimento de novos anúncios da fila.

As várias possibilidades que a variável de "status" poderá assumir são as mesmas da primitiva *recebe não bloqueante* com detecção de falhas acrescidas com estouro de tempo da primitiva bloqueante.

```
tempo da primitiva bloqueante ← valor inicial;
status ← insucesso;
enquanto status for diferente de sucesso e tempo da primitiva
         bloqueante não acabou
    faça
    início
    [ RECEBE NÃO BLOQUEANTE ]
    fim
```

ALGORÍTMO VIII.2 - IMPLEMENTAÇÃO DA PRIMITIVA "RECEBE BLOQUEANTE" USANDO A PRIMITIVA "RECEBE NÃO BLOQUEANTE"

VIII.3.3 - PRIMITIVAS COM RECUPERAÇÃO DE FALHAS

As primitivas com recuperação de falhas caracterizam-se por realizar várias tentativas de suas operações sempre que alguma anormalidade for detectada. Se a anormalidade continuar persistindo, então a execução da primitiva será terminada. Estas tentativas denominadas aqui por insistências consistirão da repetição de operações que, depois de executadas, não tenham apresentado o efeito adequado devido a atrasos inesperados, falhas explícitas, ou ações previstas no protocolo como reconhecimento negativo. As tentativas ocorrerão durante um número de vezes limitado, a partir do que o insucesso será reconhecido e assimilado pela primitiva, implicando no seu término. A recuperação ocorrá, portanto, somente quando as causas da anormalidade forem transitórias. A devolução de pacotes, indicando a presença de falhas explícitas ao longo da rede, só será detectada em estados

intermediários, uma vez que o estado inicial passará automaticamente para um estado intermediário e o estado final provocará a saída automática da primitiva.

Os tempos envolvidos nas saídas por tempo e o número de vezes de cada tentativa poderão ser definidos pelo usuário ou ajustados a nível de núcleo. O sistema poderá usar valores padrão para esses parâmetros, a menos dos tempos que dependam do processador operador, os quais poderão ficar por conta do usuário.

Após o término normal ou forçado de uma primitiva, uma variável de "status" estará indicando o motivo da saída.

VIII.3.3.1 - PRIMITIVA *ENVIA BLOQUEANTE* COM RECUPERAÇÃO DE FALHAS

No processador operador, a primitiva *envia bloqueante* com recuperação de falhas deverá funcionar conforme o diagrama de estados da figura (VIII.5).

Depois que a execução da primitiva terminar, a variável de "status" poderá estar indicando uma dentre as seguintes possibilidades:

- estouro de tempo na m-ésima vez da espera pelo reconhecimento de anúncio;
- reconhecimento negativo do anúncio na m-ésima vez;
- estouro de tempo na m-ésima vez de espera pela autorização da mensagem;
- estouro de tempo da espera pelo reconhecimento da mensagem;
- reconhecimento negativo da mensagem na m-ésima vez;
- falha explícita na rede (devolução);
- sucesso (recebimento do reconhecimento positivo de mensagem).

A primitiva será bloqueante enquanto as condições normais persistirem ou aparecerem anormalidades transitórias perfeitamente contornáveis na fase de insistência das operações de transição. Quando uma anormalidade persistir além do limite

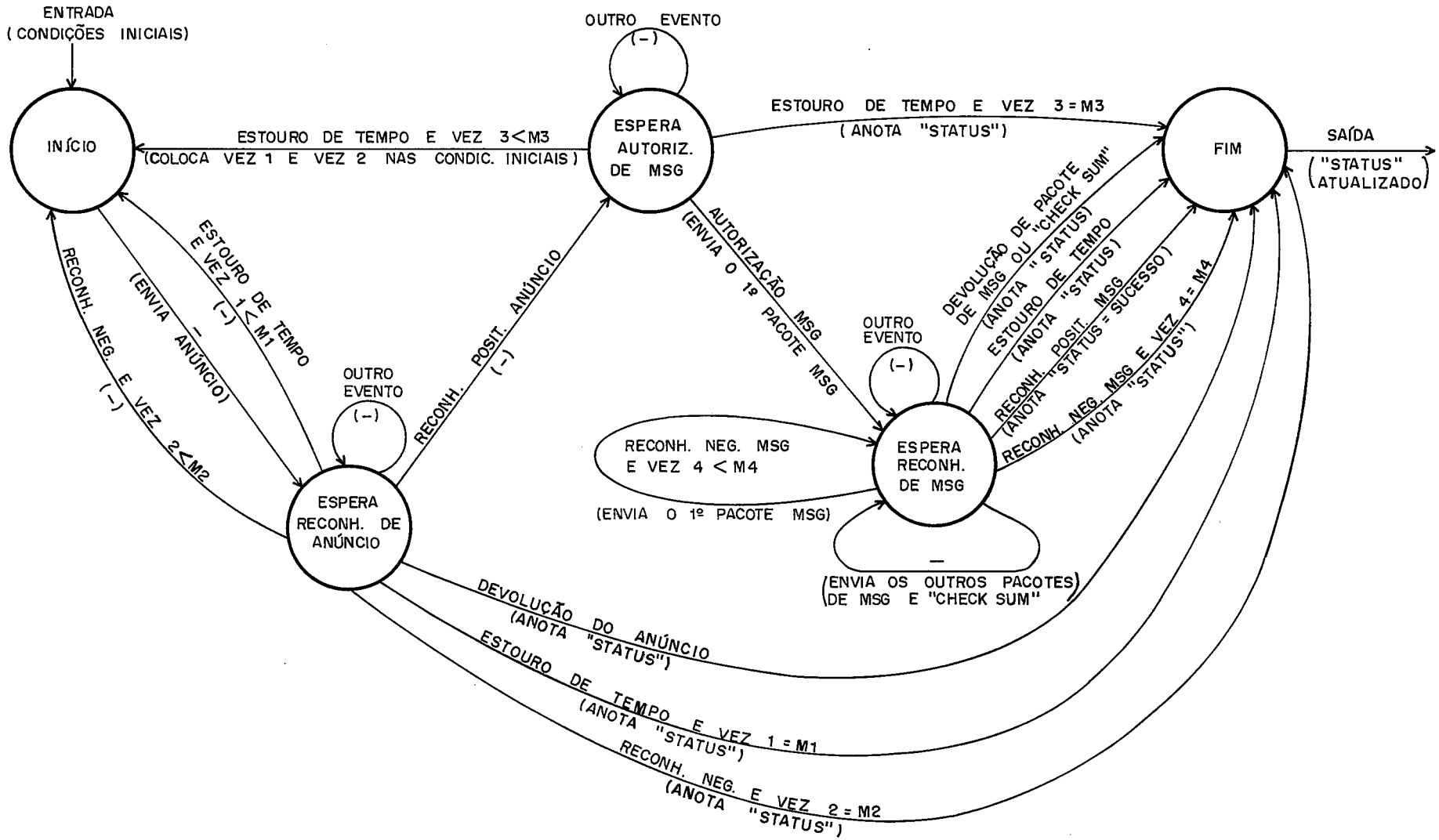


FIGURA VIII . 5 - DIAGRAMA DE ESTADOS DA PRIMITIVA "ENVIA BLOQUEANTE" COM RECUPERAÇÃO DE FALHA.

tolerado, a primitiva terminará a sua execução, e a variável de "status" indicará a causa do término forçado.

Para evitar que a eventual repetição da emissão de anúncios, implicando na colocação repetida desses anúncios na fila de anúncios, utilizou-se um número de sequência cíclico para as emissões diferentes. Quando um anúncio chegar ao processador de comunicação destino, o supervisor fará uma verificação na fila de anúncios, procurando pelo mesmo. Se encontrá-lo, haverá somente a emissão do respectivo reconhecimento, caso contrário, ele será colocado na fila e o reconhecimento será emitido. A insistência na emissão de anúncios será caracterizada pelo uso do mesmo número de sequência.

Alguns pacotes de controle obtidos com base no anúncio deverão apresentar o mesmo número de sequência do anúncio nas operações previstas pelo protocolo. Caso o número de sequência não coincida, o pacote será considerado como outro evento, sendo portanto descarregado.

VIII.3.3.2 - PRIMITIVA *RECEBE NÃO BLOQUEANTE* COM RECUPERAÇÃO DE FALHAS

No processador operador, a primitiva *recebe não bloqueante* com recuperação de falhas deverá funcionar de acordo com o diagrama de estados da figura (VIII.6).

As várias possibilidades que a variável de "status" poderá assumir após o término da primitiva são as seguintes:

- estouro de tempo na m-ésima vez da espera pela resposta;
- fila vazia;
- emissão de reconhecimento negativo de mensagem na m-ésima vez;
- estouro de tempo da espera pelos pacotes de mensagem;
- falha explícita na rede (devolução);
- sucesso (emissão de reconhecimento positivo de mensagem)..

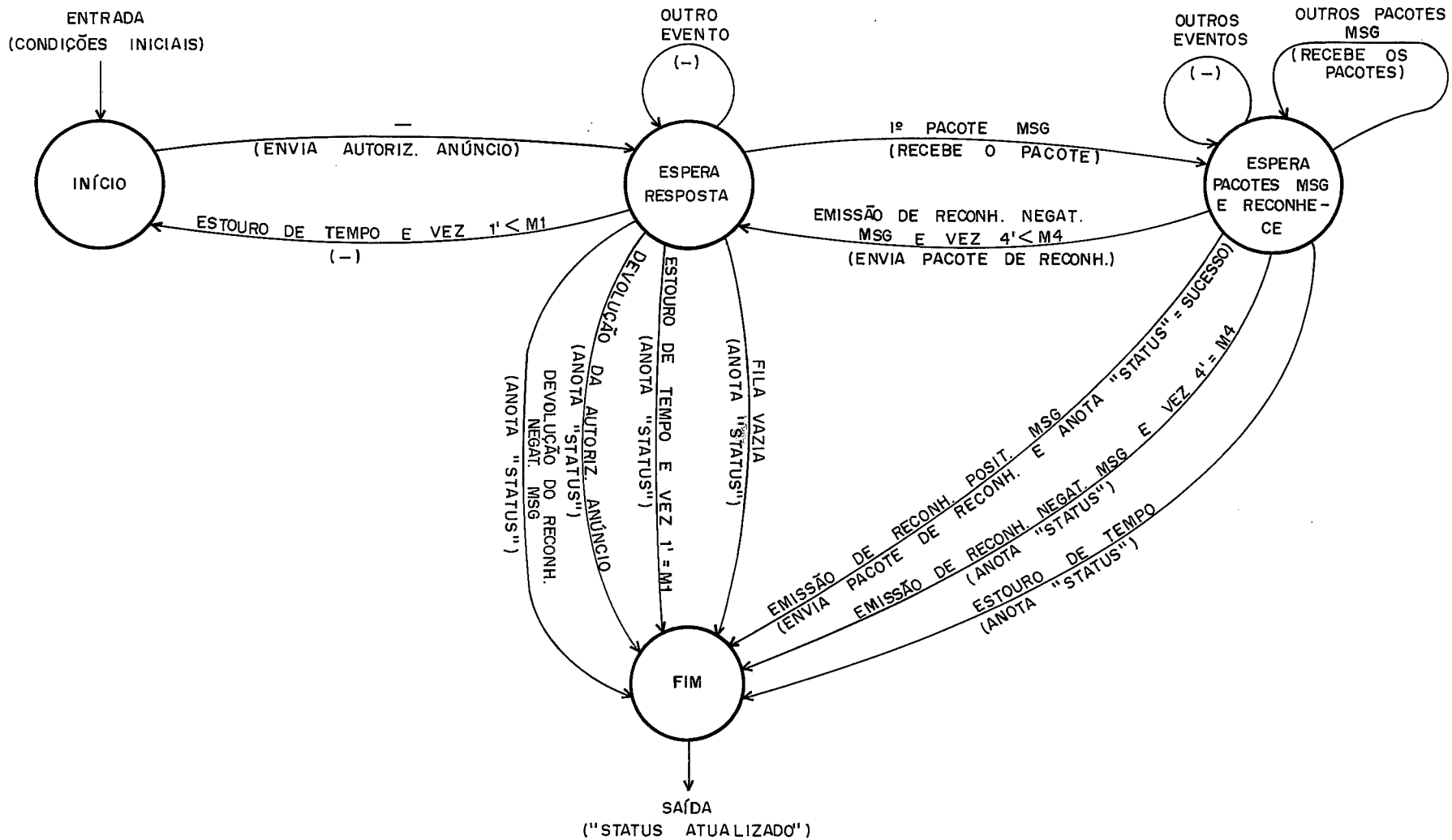


FIGURA VIII . 6 - DIAGRAMA DE ESTADOS DA PRIMITIVA "RECEBE NÃO BLOQUEANTE" COM RECUPERAÇÃO DE FALHA.

A possibilidade de recebimento da mesma autorização de anúncios por várias vezes, devido a alguma anormalidade, exigirá a manipulação do número de sequência no sentido de manter o mesmo anúncio em atendimento, quando for o caso de insistência, ou de selecionar um novo anúncio, no caso de autorização de anúncio com novo número de sequência.

A ocorrência de anormalidades verificadas na fase de recepção dos pacotes também serão objeto de insistência.

Para uma situação de funcionamento normal, as duas únicas saídas possíveis serão aquelas cuja variável de "status" contenha sucesso ou fila vazia, e o comportamento da primitiva será o mesmo para o caso da primitiva com detecção de falha.

VIII.3.3.3 - PRIMITIVA *RECEBE BLOQUEANTE* COM RECUPERAÇÃO DE FALHAS

Da mesma maneira que a primitiva para o caso de detecção de falha, esta primitiva poderá ser implementada dentro do núcleo com o uso da primitiva *recebe não bloqueante* com recuperação de falhas inserida num ciclo de repetição idêntico ao do algorítmo (VIII.2).

Como a insistência já foi considerada e o suporte necessário já existe, a repetição da primitiva não acarretará diferenças sensíveis com relação à primitiva não bloqueante.

As várias possibilidades que a variável de "status" poderá assumir são as mesmas da primitiva *recebe não bloqueante* com recuperação de falhas, acrescidas com estouro de tempo da primitiva bloqueante.

VIII.3.4 - CONSIDERAÇÕES SOBRE A PRIMITIVA *ENVIA NÃO BLOQUEANTE*

Em qualquer das duas situações analisadas, ou seja, detecção de falhas e recuperação de falhas, a implementação da primitiva *envia não bloqueante* apresentará alterações estruturais mais profundas em comparação com as primitivas vistas até agora.

O fato de não ser bloqueante faz com que o processo envie mensagens sem prévia autorização, descaracterizando, pelo me

nos em princípio, o uso do anúncio. Além disso, um processo poderá executar uma cadeia de primitivas *envia não bloqueante*, ou repetir várias vezes a execução de um laço contendo esta primitiva, podendo chegar à saturação da capacidade de armazenamento de mensagens.

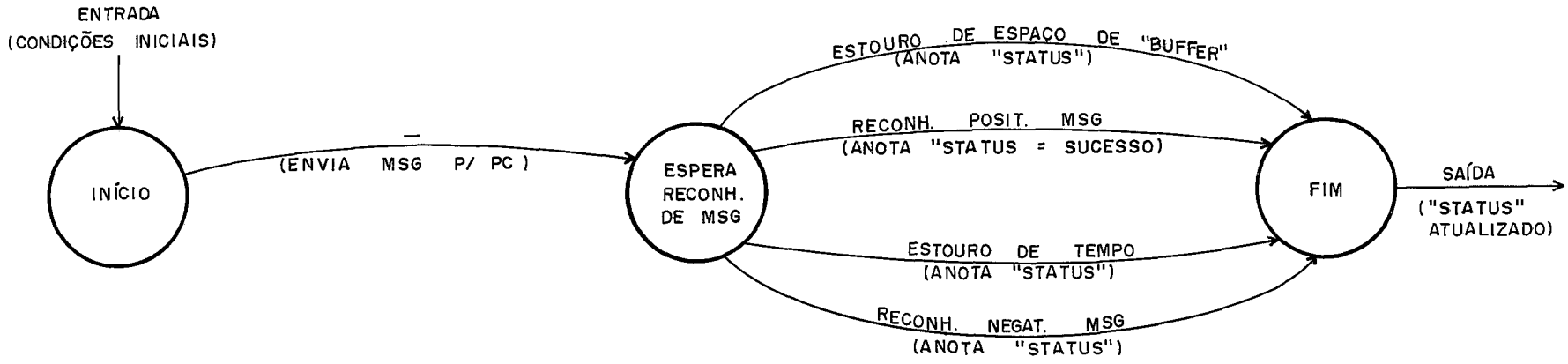
Para não alterar a filosofia de atuação adotada até agora, optou-se por considerar que o encaminhamento da mensagem ao núcleo seria a condição para a liberação da primitiva *envia não bloqueante* ao nível do processador, enquanto o processador de comunicação associado passaria a cuidar da entrega da mensagem ao seu destino. Nesse sentido, as mensagens seriam mantidas no processador de comunicação origem e seus anúncios correspondentes seriam encaminhados aos processadores de comunicação destino para serem colocados nas filas de anúncio. Assim, a execução de várias primitivas *envia não bloqueante*, situadas num processo em execução no processador operador, poderá ocorrer de forma não bloqueante, enquanto houver possibilidade de armazenamento das mensagens no processador de comunicação local. Quando não houver mais espaço, a tentativa de execução da primitiva *envia* será bloqueada à espera de espaço para a sua mensagem.

O processador de comunicação origem, recebendo uma mensagem para ser enviada, a colocará num "buffer" apropriado e passará a executar a primitiva *envia bloqueante*, encaminhando um anúncio para o processador de comunicação destino. Depois disso, o processador de comunicação origem estará esperando reconhecimento de anúncio ou nova mensagem.

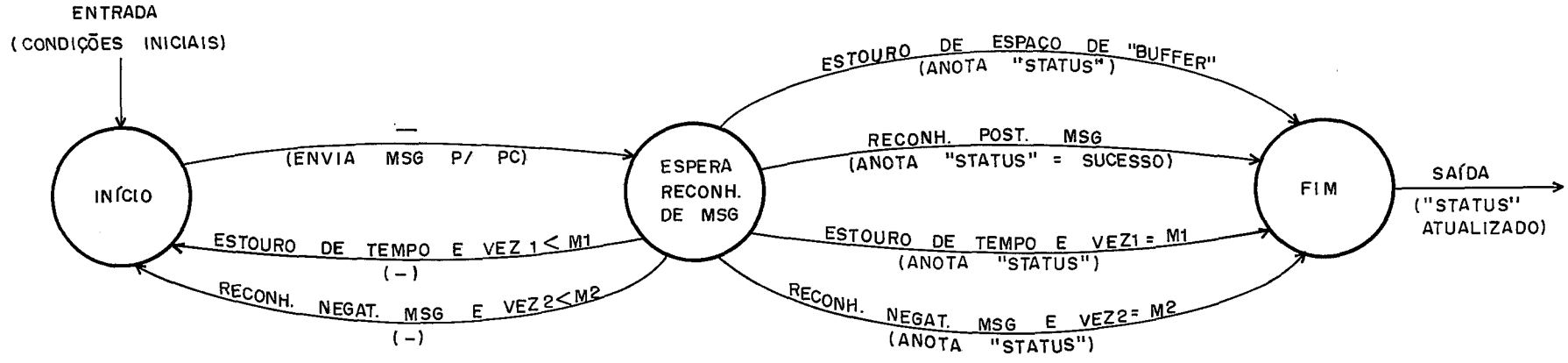
Como um único processador de comunicação poderá estar tratando da emissão de várias mensagens ao mesmo tempo, ele deverá usar rótulos atribuídos a cada primitiva, que poderão ser números de sequência, variando num intervalo maior do que a capacidade de armazenamento do processador de comunicação, em termos de número de "buffers" de mensagem. Desta maneira, o processador de comunicação estará percorrendo ao mesmo tempo vários diagramas de estado do tipo da figura (VIII.5), selecionados em cada momento pelo rótulo associado ao evento.

Do lado do processador operador, a execução da primitiva *envia não bloqueante* seguirá um dos diagramas de estado da figura (VIII.7).

De acordo com os diagramas de estado, a execução da pri



a) PRIMITIVA "ENVIA NÃO BLOQUEANTE" COM DETECÇÃO DE FALHAS.



b) PRIMITIVA "ENVIA NÃO BLOQUEANTE" COM RECUPERAÇÃO DE FALHAS.

FIGURA VIII. 7 - DIAGRAMAS DE ESTADOS DA PRIMITIVA "ENVIA NÃO BLOQUEANTE"

mitiva *envia não bloqueante* terminará, apresentando sucesso ou insucesso. O sucesso ocorrerá quando a mensagem tiver sido absorvida pelo processador de comunicação com êxito. O insucesso ocorrerá, se houver estouro no espaço de "buffers", reconhecimento negativo de mensagem, ou estouro de tempo. A primitiva *envia não bloqueante* com recuperação de falhas terá a chance de realizar algumas repetições de operações de transição, visando tentar a superação de algumas anormalidades.

Aproveitando-se o fato da maior parte do código dessa primitiva situar-se no processador de comunicação, poder-se-ia transferir para ele a maior parte do código das outras primitivas de comunicação, deixando para o processador operador somente a chamada das primitivas e a transmissão ou recebimento das mensagens. Esta atitude, por um lado facilitaria a conexão de máquinas diferentes na rede, exigindo do processador operador a incorporação de uma parcela de software pequena e simples, mas por outro lado exigiria que o processador de comunicação tivesse maior capacidade.

Além disso, o tipo de solução adotada para esta primitiva, permitindo o controle simultâneo de várias primitivas emitidas por um processo, poderá ser estendido a uma situação de multiprogramação, onde vários processos poderão disparar a execução de uma ou mais primitivas. Para isto, a estrutura lógica do processador de comunicação deverá ser ampliada convenientemente para suportar uma situação mais geral.

A primitiva *envia não bloqueante* será a base da implementação de primitivas de emissão múltipla e de difusão.

O funcionamento da rede em laço e das primitivas pode ser verificado por simulação antes da fase de implementação. Nesse sentido, consta no apêndice F as considerações sobre a simulação da rede em laço com conexão ponto a ponto.

CAPÍTULO IX

CONCLUSÕES

Este trabalho abordou a análise e o projeto de suporte para sistemas operacionais distribuídos, consistindo de um subsistema de comunicação e do software básico de comunicação. Com esta estrutura, o desenvolvimento de sistemas operacionais distribuídos pode ser realizado sem a preocupação com os detalhes de comunicação e da rede.

Para poder tratar de todos os elementos envolvidos na definição e projeto do suporte básico de sistemas operacionais distribuídos, foi necessário realizar uma revisão bibliográfica extensa de forma a abranger os vários assuntos de hardware e de software interligados com o tema.

A escolha dos dois tipos de subsistema de comunicação deveu-se ao fato deles cobrirem uma vasta gama de aplicações, permitindo a elaboração de bancadas de trabalho para desenvolvimento de sistemas operacionais distribuídos com características distintas.

O subsistema de comunicação com conexão ponto a ponto apresenta taxas de transferência que, por natureza, são relativamente baixas, e deixa todas as questões de tolerância a falhas para serem tratadas e contornadas a nível de software. Por outro lado, o subsistema de comunicação com barramento paralelo centralizado alcança taxas de transferência relativamente altas, e possui infraestrutura de hardware para realizar tratamento e superação de certas falhas, ficando outras para serem resolvidas a nível de software. Cabe salientar que, enquanto o subsistema de comunicação com conexão ponto a ponto implementa uma estrutura típica de rede, o subsistema de comunicação com barramento paralelo centralizado aproxima-se fisicamente da estrutura de multiprocessador, embora o hardware tenha sido elaborado com técnicas de distribuição.

A utilização de uma rede com barramento paralelo centra

lizado em sistemas distribuídos só é possível, devido à existência de mecanismos eficientes de tolerância a falhas, principalmente no barramento, contornando a vulnerabilidade do sistema. Além disso, como o subsistema de comunicação com barramento paralelo centralizado baseia-se em canais de difusão, os mecanismos de difusão apresentam certa aderência ao subsistema, permitindo a implementação simples e eficiente das primitivas de difusão e similares.

Levando em conta principalmente suas velocidades mais altas, o subsistema de comunicação com barramento paralelo centralizado pode ser considerado como uma boa opção para atuar como elemento de interligação de redes.

Do ponto de vista de comportamento, os dois subsistemas de comunicação apresentam detalhes característicos que podem ser explorados cientificamente: o subsistema com conexão ponto a ponto dá margem a estudos de roteamento, de controle de fluxo e de congestionamento, etc.; o subsistema com barramento paralelo centralizado propicia melhores condições de estudo de tolerância a falhas, pois o barramento e suas ligações são elementos de alta responsabilidade para o funcionamento do conjunto. A expansão do barramento paralelo centralizado e sua utilização em sistemas de aplicação em tempo real também constituem aspectos de alto interesse.

No projeto de um suporte básico para desenvolvimento de sistemas operacionais distribuídos, optou-se pela utilização de um subsistema de comunicação com conexão ponto a ponto, formando uma rede em laço. Resolveu-se também investir no desenvolvimento de uma infraestrutura de software, contendo mecanismos adequados para o tratamento e recuperação automática de falhas.

Nesse suporte, a superação de falhas transitórias ocorrerá, sempre que for necessário, por meio de procedimentos de insistência embutidos nas primitivas de comunicação. Tais procedimentos farão com que operações mal sucedidas sejam repetidas um certo número de vezes, antes de determinar insucesso. A superação de falhas permanentes, por sua vez, dar-se-á através da mudança automática de rota, isolando automaticamente os elementos que falharem e reintegrando-os, também automaticamente, quando voltarem ao funcionamento normal.

Vários ítens do projeto, como formato dos pacotes, pro

grama supervisor, protocolo de comunicação usando anúncio, primitivas de comunicação, etc., foram estabelecidos de forma integrada pelo fato de exercerem influência entre si.

O protocolo de comunicação entre processadores operadores usando anúncio apresenta uma certa complexidade para a atividade de troca de mensagens e, em princípio, uma sobrecarga adicional. Em compensação, o uso do protocolo, mantendo os "buffers" na origem, proporciona uma comunicação mais justa, pois evita que um processo possa interferir no uso de "buffers" localizados em outros processadores. Como o anúncio é de tamanho reduzido, admite-se que a fila de anúncios sempre tenha capacidade para armazenar todos os anúncios correspondentes a todos os "buffers" do sistema. Desta forma, cada processo poderá usar os "buffers", conforme sua capacidade local.

A questão da sobrecarga decorrente do protocolo é discutível, pois se por um lado é necessário que o anúncio seja remetido para o destino, seguido de uma fase de espera pela autorização para um processo poder enviar uma mensagem, por outro lado tem-se a certeza de que, em condições normais, as mensagens terão garantias de recebimento sob qualquer situação de carga. A alternativa de uso de "buffers" no destino apresenta maior rapidez em situações de carga baixa, quando sempre existem "buffers" disponíveis. Entretanto, esta alternativa sofre uma degradação considerável, quando a carga aumenta e as mensagens passam a ser perdidas por falta de espaço, exigindo a sua repetição e aumentando ainda mais a sobrecarga do subsistema de comunicação.

A existência de uma fila de anúncios no processador de comunicação destino permite que o processo possa trabalhar com primitivas *recebe*, envolvendo vários tipos de endereçamento. O comando *autoriza anúncio* pode autorizar o atendimento de qualquer anúncio, ou solicitar a seleção de um anúncio específico baseada em endereço, prioridade, operação, etc., facilitando a estruturação do sistema.

Como as primitivas de comunicação foram projetadas para funcionar num ambiente passível de falhas, previu-se o aparecimento de pacotes de controle e de informação órfãos, devido ao desaparecimento ou isolação de determinados nós da rede, depois de terem emitidos tais pacotes. Previu-se também o aparecimento de pacotes repetidos, gerados como consequência da não emissão

ou perda de reconhecimento. Para contornar essas questões, usou-se número de sequência, tempo limite para saída ("time-out"), devolução de pacotes, e outros mecanismos que permitissem o tratamento adequado de cada caso.

No sentido de dar continuidade ao trabalho desenvolvido, poder-se-ia atuar na melhoria das bancadas de trabalho e na sua utilização em desenvolvimento de sistemas. Assim, uma das principais providências que seriam tomadas consistiria em aumentar o desempenho do hardware das bancadas, através da introdução de circuitos de acesso direto à memória (DMA), elevando as taxas de transferência dos processadores de comunicação. A implantação de circuitos adicionais, constituindo tanto "buffers" de apoio a estruturas com prioridades, quanto "buffers" de difusão, que evitariam a perda de mensagens quando ocorresse difusão, melhorariam os processadores de comunicação ligados ao barramento paralelo centralizado.

O suporte completo usando um subsistema de comunicação com barramento paralelo centralizado poderia ser desenvolvido e o suporte com conexão ponto a ponto poderia tornar-se mais flexível pela transformação da rede em laço em rede irregular. Isto exigiria uma adaptação no algorítmo de rota e nos procedimentos de isolação e reintegração automáticas.

O subsistema de comunicação com barramento paralelo centralizado poderia ter sua taxa de tolerância a falhas aumentada, visando a sua aplicação em tempo real.

Algumas primitivas de comunicação de nível mais alto, como chamada remota de procedimento e "rendez-vous" estendido, poderiam ser desenvolvidas.

As duas bancadas de trabalho poderiam ser interconectadas, permitindo o desenvolvimento de mecanismos de comunicação entre as mesmas.

Finalmente, dentre outras coisas, poderia ser desenvolvido um sistema operacional distribuído com reconfiguração dinâmica, usando a bancada de trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

1 - ACAMPORA, A. & HLUCHYJ, M.G. - "A New Local Area Network Architecture Using a Centralized Bus". *IEEE Communications Magazine*, 22(8):12-21, aug. 1984.

2 - AHAMAD, M. & BERNSTEIN, A.J. - "An Application of Name Based Addressing to Low Level Distributed Algorithms". *IEEE Transactions on Software Engineering*. SE-11(1): 59-67, jan. 1985.

3 - AMES JR., S.R. & OESTREICHER, D.R. - "Design of a Message Processing System for a Multilevel Secure Environment". In: *Proc. of National Computer Conference 78*. 1978. p. 765-771.

4 - ANDERSON, G.A. & JENSEN, E.D. - "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples". *ACM Computing Surveys*, 7(4):197-213, dec. 1975.

5 - ANDERSON, T. & KNIGHT - *Practical Software Fault Tolerance for Real-Time Systems*. Technical Report Series, 169, Computing Laboratory, University of Newcastle Upon Tyne, England, jun. 1981. 40 p.

6 - ANDREWS, G.R. - "Synchronizing Resources". *ACM Transactions on Programming Languages and Systems*, 3(4):405-430, oct. 1981.

7 - ANDREWS, G.R. - "Distributed Programming Languages". In: *Proc. of the ACM'82 Conference*, Dallas, Texas, ACM, oct. 1982. p. 113-117.

8 - ANDREWS, G.R. - "The Distributed Programming Language SR - Mechanisms, Design and Implementation". *Software - Practice and Experience*, 12(8):719-753, aug. 1982.

9 - ANDREWS, G.R. & SCHNEIDER, F.B. - "Concepts and Notations for Concurrent Programming". *ACM Computing Surveys*. 15(1):3-43, mar. 1983.

10 - APPELBE, W.F. - "A Survey of Systems Programming Languages: Concepts and Facilities". *Software - Practice and Experience*, 15(2):169-190, feb. 1985.

11 - BACON, J. - "An Approach to Distributed Software Systems". *ACM Operating Systems Review*, 15(4):62-74, oct. 1981.

12 - BARAK, A. & LITMAN, A. - "MOS: A Multicomputer Distributed Operating Systems". *Software - Practice and Experience*. 15(8):725-737, aug. 1985.

13 - BARZAK, C.L. & AZEVEDO, I.A. - "Critérios e Estratégias para o Projeto de Sistema de Computação Tolerante a Falhas". In: *1º Simpósio em Sistemas de Computadores Tolerantes a Falhas*. S.J. dos Campos, SP, INPE, set./out. 1985. p. 3-23.

14 -BASKETT, F.; HOWARD, J.H. & MONTANGUE, J.T. - "Task Communications in DEMOS". In: *Proc. of 6th Symposium on Operating Systems Principles*. West Lafayette, ACM, nov. 1977. p. 23-31.

15 - BERNSTEIN, P.A. & GOODMAN, N. - "Concurrency Control in Distributed Database Systems". *ACM Computing Surveys*, 13(2):185-221, jun. 1981.

16 - BERRY, D.M. et alii - *Language Constructs for Real-Time Distributed Systems*. Report Interno, 22, Laboratorio di Calcolatori, Istituto di Elettrotecnica ed Elettronica del Politecnico di Milano, Milano, Itália, dic. 1979. 28 p.

17 - BIRRELL, A.D. - "Secure Communication Using Remote Procedure Calls". *ACM Transactions on Computer Systems*, 3(1): 1-14, feb. 1985.

18 - BIRRELL, A.D. & NELSON, B.J. - "Implementing Remote Procedure Calls". *ACM Transaction on Computer Systems*, 2(1): 39-59, feb. 1984.

19 - BOCHMANN, G.V. & RAYAL, M. - "Structured Specification of Communicating Systems". *IEEE Transaction on Computers*, C-32(2):120-133, feb. 1983.

20 - BRAYER, K. & LAFLEUR, V. - "A Testbed Approach to the Design of a Computer Communication Network". *Computer*, 15(10):14-23, oct. 1982.

21 - BRINCH HANSEN, P. - "Structured Multiprogramming". *Communications of the ACM*, 15(7):574-578, jul. 1972.

22 - BRINCH HANSEN, P. - *Operating System Principles*. Englewood Cliffs, N.J., Prentice-Hall, Inc., 1973. 366 p.

23 - BRINCH HANSEN, P. - "The Programming Language Concurrent Pascal". *IEEE Transaction on Software Engineering*, SE-1(2):199-207, jun. 1975.

24 - BRINCH HANSEN, P. - *The Architecture of Concurrent Programs*. Englewood Cliffs, N.J., Prentice-Hall, Inc., 1977. 317p.

25 - BRINCH HANSEN, P. - "Network: A Multiprocessor Program". *IEEE Transaction on Software Engineering*, SE-4(3):194-199, may 1978.

26 - BRINCH HANSEN, P. - "Distributed Processes: A Concurrent Programming Concepts". *Communications of the ACM*, 21(11):934-941, nov. 1978.

27 - BRINCH HANSEN, P. & FELLOWS, J. - "The Trio Operating System". *Software - Practice and Experience*, 10(11):943-948, nov. 1980.

28 - BRINCH HANSEN, P. - *Edison - A Multiprocessor Language*. *Software - Practice and Experience*, 11(4):325-361, apr. 1981.

29 - BRINCH HANSEN, P. - *Programming a Personal Computer*. Englewood Cliffs, N.J., Prentice-Hall, Inc., 1982. 388 p.

30 - BROWN, R.L.; DENNING, P. & TICHY, W.F.-"Advanced Operating Systems". *Computer*, 17(10):173-190, oct. 1984.

31 - BROWN, R.R. - "MCTS Nucleus: Philosophy and Praxis". *ACM Operating Systems Review*, 10(1):39-60, jan. 1976.

32 - BUSTARD, D. & HOARE, C.A.R. - *A Nucleus for a Multiprocessor Multiprogramming System*. Technical Report, Dept. of Computer Science, Quen's University of Belfast, may 1975. 24 p.

33 - CARPENTER, B.E. & CAILLIAU, R. - "Experience with Remote Procedure Call in a Real-Time Control System". *Software - Practice and Experience*, 14(9):901-907, sep. 1984.

34 - CASAGLIA, G.F. - "Distributed Computing Systems: A Biased Review". *Euromicro Journal*, 2(4):5-18, oct. 1976.

35 - CHERITON, D.R. - "The Design of a Distributed Kernel". In: *Proc. of ACM'81 Conference*, Los Angeles, Califórnia, ACM, nov. 1981. p. 46-52.

36 - CHERITON, D.R. et alii - "Thoth, a Portable Real-Time Operating System". *Communications of the ACM*, 22(2):105-115, feb. 1979.

37 - COOK, R.P. - "*MOD - A Language for Distributed Programming". *IEEE Transactions on Software Engineering*, SE-6(6):563-571, nov. 1980.

38 - CORSO, D.D. & VERRUA, L. - "Contention Delay in Distributed Priority Networks". *Microprocessing and Microprogramming*, 13(1):21-29, jan. 1984.

39 - CUNHA, P.R.F.; LUCENA, C.J. & MAIBAUM, T.S.E. - "On the Design and Specification of Message Oriented Programs". *International Journal of Computer and Information Sciences*, 9(3), jun. 1980.

40 - DAVENPORT, R.A. - "Distributed Database Technology - A Survey". *Computer Networks*, 2(3):155-167, jul. 1978.

41 - DAVENPORT, R.A. - "Design of Distributed Data Base Systems". *The Computer Journal*, 24(1):31-41, feb. 1981.

42 - DAVIES, D.W. et alii - *Computer Networks and Their Protocols*. Chichester, John Wiley & Sons, 1979. 487 p.

43 - DAY, J.D. & ZIMMERMANN - "The OSI Reference Model". *Proceedings of the IEEE*, 71(12):1334-1340, dec. 1983.

44 - DEITEL, H.M. - *An Introduction to Operating Systems*. Reading, Mass., Addison-Wesley Publishing Company, 1984. 673 p.

45 - DIJKSTRA, E.W. - "The Structure of the THE - Multiprogramming System". *Communications of the ACM*, 11(5):341-346, may 1968.

46 - DONNELLEY, J. - "Components of a Network Operating System". In: *Proc. 4th Conference on Local Computer Networks*, Minneapolis, Minnesota, IEEE, oct. 1979. p. 1-12.

47 - ECKHOUSE JR., R.H. et alii - "Issues in Distributed Processing - An Overview of Two Workshops". *Computer*, 11(1):22-26, jan. 1978.

48 - ELOVITZ, H.S. & HEITMEYER, C.L. - "What is a Computer Network". In: *Proc. of National Telecommunications Conference 74*. 1974. p. 1007-1014.

49 - ENSLOW JR., P.H. ed. - *Multiprocessors and Parallel Processing*. New York, John Wiley & Sons, 1974. 340 p.

50 - ENSLOW JR., P.H. - "Multiprocessor Organization - A Survey". *ACM Computing Surveys*, 9(1):103-129, mar. 1977.

51 - ENSLOW JR., P.H. - "What is a Distributed Data Processing System". *Computer*, 11(1):13-21, jan. 1978.

52 - FANTECHI, A.; INVERARDI, P. & LUTMAER, N. - *The Cnet Inter-Node Communication Mechanism*. Public. Int. del Progetto Finalizzato Informatica C.N.R. - Progetto Cnet, 100, Pisa, Itália, dez. 1983. 26 p.

53 - FELDMAN, J.A. - "High Level Programming for Distributed Computing". *Communications of the ACM*, 22(6):353-368, jun. 1979.

54 - FINKEL, R.A. - "Issues for Distributed Operating Systems". In: *Proc. of INFOCOM 82*, Los Angeles, California, IEEE, 1982. p. 204-205.

55 - FINKEL, R.A. & SOLOMON, M.H. - *The Aracne Distributed Operating System*. Computer Sciences Technical Report, 439, Computer Sciences Department, University of Wisconsin, Madison, jul. 1981. 41 p.

56 - FRANK, A.J.; WITTIE, L.D. & BERNSTEIN, A.J. - "Multicast Communication on Network Computers". *IEEE Software*, 2(3):49-61, may 1985.

57 - GARRETTI, P.; LAFACE, P. & RIVOIRA, S. - "MOSDOK: A Modular Distributed Operating System Kernel for Real-Time Process Control". *Microprocessing and Microprogramming*, 9:201-213, 1982.

58 - GENTLEMAN, W.M. - "Message Passing Between Sequential Processes: The Reply Primitive and the Administrator Concept". *Software - Practice and Experience*, 11(5): 435-466, may 1981.

59 - GIEN, M. - "A File Transfer Protocol (FTP)". *Computer Networks*, 2(4/5):312-319, sep./oct. 1978.

60 - GRAEF, N. et alii - "How to Design and Implement Small Time-sharing Systems Using Concurrent Pascal". *Software - Practice and Experience*, 9(1):17-24, jan. 1979.

61 - GRIMSDALE, R.L. et alii - "Polyproc II - The University of Sussex Multiple Microprocessor System". In: *Proc. 2nd Int. Conf. on Distributing Computing Systems*. Versailles, France, IEEE, 1981. p. 95-104.

62 - HABERMANN, A.N.; FLON, L. & COOPRIDER, L. - "Modularization and Hierarchy in a Family of Operating Systems". *Communications of the ACM*, 19(5):266-272, may 1976.

63 - HAC, A. - "Distributed File Systems - A Survey". *ACM Operating Systems Review*, 19(1):15-18, jan. 1985.

64 - HARIDI, S.; BAUNER, J.O. & SVENSSON, G. - "An Implementation and Empirical Evaluation of the Tasking Facilities in ADA". *ACM Sigplan Notices*, 16(2):35-47, feb. 1981.

65 - HEWITT, C.E. & ATKINSON, R.R. - "Specification and Proof Techniques for Serializers". *IEEE Transactions on Software Engineering*, SE-5(1):10-23, jan. 1979.

66 - HOARE, C.A.R. - "Monitors: An Operating System Structuring Concept". *Communications of the ACM*, 17(10):549-557, oct. 1974.

67 - HOARE, C.A.R. - "Communicating Sequential Process". *Communications of the ACM*, 21(8):666-677, aug. 1978.

68 - HOLT, R.C. - *Concurrent Euclid, The UNIX System, and TUNIS*. Reading, Mass., Addison-Wesley Publishing Company,1983. 323 p.

69 - HOLT, R.C. et alii - *Structured Concurrent Programming with Operating Systems Applications*. Reading, Mass., Addison-Wesley Publishing Company, 1978. 262 p.

70 - HOPPE, J. - "A Simple Nucleus Written in Modula-2: A Case Study". *Software - Practice and Experience*, 10(9):697-706, sep. 1980.

71 - ICHBIAH, J.D. et alii - "Rationale for the Design of the ADA Programming Language". *ACM SIGPLAN Notices*, 14(6): Part B, jun. 1979.

72 - JAMEL, A.J. & STIEGLER, H.G. - "Managers Versus Monitors". In: *Proc. of IFIP Congress 77*, Toronto, North-Holland Publishing Company, aug. 1977. p. 827-830.

73 - JONES, A.K. - "The Narrowing Gap Between Language Systems and Operating Systems". In: *Proc. of IFIP Congress 77*, Toronto, North-Holland Publishing Company, aug. 1977. p. 869-873.

74 - JONES, A.K. - "The Object Model: A Conceptual Tool for Structuring Software". In: BAYER, R.; GRAHAM, R.M. & SEEGMULLER, G. (eds.) - *Operating Systems: An Advanced Course*. Berlin, Springer-Verlag, 1978. p. 7-16. (Lecture Notes in Computer Science, 60).

75 - JONES, A.K. & SCHWARZ, P. - "Experience Using Multiprocessor Systems - A Status Report". *ACM Computing Surveys*, 12(2):121-165, jun. 1980.

76 - JONES, A.K. et alii - "Star OS, a Multiprocessor Operating System for the Support of Task Forces". In: *Proc. 7th Symposium on Operating Systems Principles*. Pacific Grove, N.Y., ACM, dec. 1979. p. 117-127.

77 - JOSEPH, M. - *Schemes for Communication*. Technical Report, 122, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, jun. 1981. 41 p.

78 - JOSEPH, M.; PRASAD, V.R. & NATARAJAN, N. - *A Multiprocessor Operating System*. Englewood Cliffs, N.J., Prentice-Hall, Inc., 1984. 477 p.

79 - KAMIBAYASHI, N. et alii - "HEART: An Operating Systems System Nucleus Machine Implemented by Firmware". In: *Proc. of Symposium on Architectural Support for Programming Languages and Operating Systems*. Palo Alto, Califórnia, ACM, apr. 1982. p. 195-204.

80 - KATAGUIRI, E.K. - "Alternativas para Comunicação entre Processos em Sistemas Distribuídos". *Boletim SCOPUS*, (67): 1-12, jan. 1984.

81 - KATZAN JR., H. - *An Introduction to Distributed Data Processing*. N.Y., Petrocelli Book, 1978. 242 p.

82 - KEEDY, J.L. - "On Structuring Operating Systems with Monitors". *The Australian Computer Journal*, 10(1):5-9, feb. 1978.

83 - KEPECS, J. & SOLOMON, M. - "SODA: A Simplified Operating System for Distributed Applications". *ACM Operating Systems Review*, 19(4):45-56, oct. 1985.

84 - KIRNER, C. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Uma implementação". In: *Anais do 4º Congresso da Sociedade Brasileira de Computação - 11º SEMISH*. Viçosa, MG, SBC, jul. 1984. p. 109-121.

85 - KIRNER, C. & MARQUES, E. - "Suporte para Desenvolvimento de Sistemas Distribuídos Baseado num Barramento Paralelo Centralizado". In: *Anais do 5º Congresso da Sociedade Brasileira de Computação - 11ª CLAI - 12º SEMISH*, Porto Alegre, RS, SBC, jul. 1985. p. 423-435.

86 - KIRNER, C. & SANTOS, S.M. - "Projeto de Máquina Básica para a Implantação de Sistemas Operacionais num Computador de Arquitetura Clepsidra". In: *Anais do 8º Seminário Integrado de Software e Hardware*. Florianópolis, SC, SBC, jul. 1981. p. 47-58.

87 - KIRNER, C.; SEGRE, L.M. & SANTOS, S.M. - "Evolução da Proposta da Linguagem para Programação Concorrente LPC". In: *Anais do 4º Congresso da Sociedade Brasileira de Computação - 11º SEMISH*. Viçosa, MG, SBC; jul. 1984. p. 477-479.

88 - KLEINROCK, L. - "Distributed Systems". *Communications of the ACM*, 28(11):1200-1213, nov. 1985.

89 - KOCH, A. & MAIBAUM, T.S.E. - *MENYMA: A General Purpose Message Oriented Language*. Relatório Técnico, Departamento de Informática, Pontifícia Universidade Católica, Rio de Janeiro, RJ, 1981. 32 p.

90 - KOHLER, W.H. - "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems". *ACM Computing Surveys*, 13(2):149-183, jun. 1981.

91 - KRAMER, J. et alii - "CONIC - An Integrated Approach to Distributed Computer Control Systems". *IEE Proc.*, 130(1):1-10, jan. 1983.

92 - KRAMER, J.; MAGEE, J. & SLOMAN, M. - "Intertask Communication Primitives for Distributed Computer Control System". In: *Proc. of 2nd Int. Conf. Distributed Computing Systems*, Paris, ACM, apr. 1981. p. 404-411.

93 - KRONENTAL, M. - "Towards the Standardization of Real-Time Operating System Kernels". In: *Proc. of the 2nd IFAC/IFIP Symposium on Software for Computer Control*, Prague, Czechoslovakia, Pergamon Press, jun. 1979. p. 181-188.

94 - KUROSE, J.F.; SCHWARTZ, M. & YEMINI, Y. - "Multiple-Access Protocols and Time-Constrained Communication". *ACM Computing Surveys*, 16(1):43-70, mar. 1984.

95 - KUTTI, S. - "Why a Distributed Kernel?". *ACM Operating Systems Review*, 18(4):5-11, oct. 1984.

96 - LAMPSON, B.W. & REDELL, D.D. - "Experience with Processes and Monitors in Mesa". *Communications of the ACM*, 23(2):105-117, feb. 1980.

97 - LAMPSON, B.W. & STURGIS, H.E. - "Reflections on an Operating System Design". *Communications of the ACM*, 19(5): 251-265, may 1976.

98 - LAUER, H.C. & NEEDHAM, R.M. - "On the Duality of Operating Systems Structures". In: *Proc. of 2nd Int. Symposium on Operating Systems*. IRIA, oct. 1978, reimpresso: ACM Operating System Review, 13(2):3-19, apr. 1979.

99 - LE LANN, G. - "Distributed Systems - Towards a Formal Approach". In: *Proc. of IFIP Congress 77*, Toronto, North-Holland Publishing Company, aug. 1977. p. 155-160.

100 - LEBLANC, T.J.; GERBER, R.H. & COOK, R. - "The StarMod Distributed Programming Kernel". *Software - Practice and Experience*, 14(12):1123-1139, dec. 1984.

101 - LI, C.M. & LIU, M.T. - "Dislang: A Distributed Programming Language/System". In: *Proc. of 2nd Int. Conf. Distributed Computing Systems*. Paris, IEEE, apr. 1981. p. 162-172.

102 - LINDSAY, D.C. - "Local Area Networks: Bus and Ring Vs. Coincident Star". *Computer Communications Review*, 12(3,4):83-91, jul./oct. 1982.

103 - LISKOV, B. - "Primitives for Distributed Computing". In: *Proc. of 7th Symposium on Operating Systems Principles*. Pacific Grove, N.Y., ACM, dec. 1979. p. 33-42.

104 - LISKOV, B. - *Remote Procedure Call*. Technical Report, DSG Note 64, Laboratory for Computer Science, M.I.T., Mass., jun. 1980. 6 p.

105 - LISKOV, B. - "On Linguistic Support for Distributed Programs". *IEEE Transactions on Software Engineering*, SE-8(3):203-210, may 1982.

106 - LIU, M.T. & LI, C.M. - "Communicating Distributed Process: A Language Concept for Distributed Programming in Local Area Networks". In: WEST, A. & JANSON, P. (eds.) - *Local Networks for Computer Communications*. Amsterdam, North-Holland Publishing Company, 1981. p. 375-406.

107 - LORIN, H. - "Distributed Processing: An Assessment". *IBM Systems Journal*, 18(4):582-603, apr. 1979.

108 - LORIN, H. & DEITEL, H.M. - *Operating Systems*. Reading, Mass., Addison-Wesley Publishing Company, 1981. 378 p.

109 - MAGEE, J.N. - *Provision of Flexibility in Distributed Systems*. Ph.D. Thesis, Dept. of Computing, Imperial College, London, apr. 1984. 135 p.

110 - MANNING, E.; LIVESEY, N.J. & TOKUDA, H. - "Interprocess Communication in Distributed Systems: One View". In: *Proc. of IFIP Congress 80*, North-Holland Publishing Company, 1980. p. 513-520.

111 - MAO, T.W. & YEH, R. - "Communication Port: A Language Concept for Concurrent Programming". *IEEE Transaction on Software Engineering*, SE-6(2):194-204, mar. 1980.

112 - MARQUES, E. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Implementação do Software Básico de Comunicação". In: *Anais do IV Simpósio sobre Desenvolvimento de Software Básico*. S.J. dos Campos, SP, ITA, out. 1984. p. 184-196.

113 - MARTELLI, M. & TARINI, F. - *Meccanismi di Comunicazione "Inter-Node" (Analisi e Proposte per una Lista di Requisiti)*. Public. Int. del Progetto Finalizzato Informatica C.N.R. - Progetto Cnet, 15, Pisa, Itália, nov. 1980. 23 p.

114 - MARTIN, J. - *Computer Networks and Distributed Processing - Software, Techniques, and Architecture*. Englewood Cliffs, N.J., Prentice-Hall, 1981. 562 p.

115 - McKENZIE, A.A. et alii - "The ARPA Network Control Center". In: *Proc. of 4th Data Communications Symposium*. oct. 1975. p. 5.1-5.6.

116 - McNAMARA, J.E. - *Technical Aspects of Data Communication*. Bedford, Mass., Digital Press, 1977. 387 p.

117 - MILLEN, J.K. - "Security Kernel Validation in Practice". *Communication of the ACM*, 19(5):243-250, may 1976.

118 - MORON, C.E. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Supervisor do Processador de Comunicação". In: *Anais do 4º Congresso da Sociedade Brasileira de Computação - 11º SEMISH*. Viçosa, MG, SBC, jul. 1984. p. 123-131.

119 - MUELLER, E.T.; MOORE, J.D. & POPEK, G.J. - "A Nested Transaction Mechanism for LOCUS". In: *Proc. of 9th Symposium on Operating Systems Principles*. Bretton Woods, New Hampshire, ACM, oct. 1983. p. 71-89.

120 - NATARAJAN, N. & SINHA, M.K. - "Language Issues in the Implementation of a Kernel". *Software - Practice and Experience*, 9(9):771-778, sep. 1979.

121 - NELSON, B.J. - *Remote Procedure Call*. Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon University, may 1981. 201 p. (Report CMU-CS-81-119).

122 - OUSTERHOUT, J.K. et alii - "Medusa: An Experiment in Distributed Operating System Structure". *Communications of the ACM*, 23(2):92-105, feb. 1980.

123 - PARDO, R. - *Interprocess Communication and Synchronization for Distributed Systems*. Ph.D. Thesis, Dept. of Computer And Information Science, The Ohio State University, 1979. 229 p.

124 - PARNAS, D.L. - "On the Criteria to be Used in Decomposition into Modules". *Communication of the ACM*, 15(12):1053-1058, dec. 1972.

125 - PAUL, M. & SIEGERT, H.J. - *Distributed Systems - Methods and Tools for Specification: An Advanced Course*. Berlin, Springer-Verlag, 1985. 573 p. (Lecture Notes in Computer Science, 190).

126 - PEEBLES, R. & MANNING, E. - "System Architecture for Distributed Data Management". *Computer*, 11(1):40-47, jan. 1978.

127 - PETERSON, J. - "Notes on a Workshop on Distributing Computing". *ACM Operating Systems Review*, 13(3):18-30, jul. 1979.

128 - POPEK, G.J. & FARBER, D.A. - "A Model for Verification of Data Security in Operating Systems". *Communications of the ACM*, 21(9):737-749, sep. 1978.

129 - POPEK, G.J. & KLINE, C.S. - "A Verifiable Protection System". In: *Proc. of Int. Conf. Reliable Software*, Los Angeles, Califórnia, apr. 1975. p. 294-304.

130 - POPEK, G.J. & KLINE, C.S. - "Issues in Kernel Design". In: BAYER, R.; GRAHAM, R.M. & SEEGMULLER, G. (eds.) - *Operating Systems: An Advanced Course*. Berlin, Springer-Verlag, 1978. p. 210-227. (Lecture Notes in Computer Science, 60).

131 - POWELL, M.L. & MILLER, B.P. - "Process Migration in DEMOS/MP". In: *Proc. of 9th Symposium on Operating Systems Principles*. Bretton Woods, New Hampshire, ACM, oct. 1983. p. 110-119.

132 - POWELI, M.S. - "Experience of Transporting and Using the SOLO Operating System". *Software - Practice and Experience*, 9(7):561-569, jul. 1979.

133 - PRASAD, K.V.S.; NATARAJAN, N. & SINHA, M.K. - "Physical and Logical Abstractions in a Kernel". In: LANCIAUX, D. (ed.) - *Operating Systems: Theory and Practice*. North-Holland Publishing Company, 1979. p. 359-368.

134 - PRICE, R.J. - "A Language for Distributed Processing". In: *Proc. of National Computer Conference 79*. 1979.

135 - RANDEL, B. - *Fault Tolerance & System Structuring*. Technical Report Series, 189, Computing Laboratory, University of Newcastle Upon Tyne, England, dec. 1983. 20 p.

136 - RASHID, R.F. - *An Inter-Process Communication Facility for Unix*. Technical Report, 24, Dep. of Computer Science, Carnegie-Mellon University, Pittsburgh, jun. 1980. 35 p.

137 - RASHID, R.F. & ROBERTSON, G.G. - "Accent: A Communication Oriented Network Operating System Kernel". In: *Proc. of 8th Symposium on Operating Systems Principles*. Pacific Grove, Califórnia, ACM, dec. 1981. p. 64-75.

138 - RAVASIO, P.C.; HOPKINS, G. & NAFFAH, N. (eds.) - *Local Computer Networks*. Amsterdan, North-Holland, 1982. 555 p.

139 - REDELL, D.D. et alii - "Pilot: An Operating System for a Personal Computer". *Communications of the ACM*, 23(2): 81-92, feb. 1980.

140 - REES, D.J.& STEPHENS, P.D. - "The Kernel of the EMAS 2900 Operating System". *Software - Practice and Experience*, 12(7):655-667, jul. 1982.

141 - REID, L.G. - *Control and Communication in Programmed Systems*. Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon University, sep. 1980. 148 p. (Tech. Rep. CMU-CS-80-142.

142 - RICHARDS, M. et alii - "TRIPOS - A Portable Operating System for Mini-computers". *Software - Practice and Experience*, 9(7):513-526, jul. 1979.

143 - RODGERS, D. - "Improvements in Multiprocessor System Design". In: *Proc. of 12th Annual International Symposium on Computer Architecture*. Boston, Mass., ACM, jun. 1985. p. 225-231.

144 - RUSHBY, J.M. - "Design and Verification of Secure Systems". In: *Proc. of 8th Symposium on Operating Systems Principles*. Pacific Grove, Califórnia, ACM, dec. 1981. p. 12-21.

145 - RUSHBY, J.M. & RANDEL, B. - *A Distributed Secure Systems*. Technical Report Series, 182, Computing Laboratory, University of Newcastle Upon Tyne, England, may 1982. 44 p.

146 - SANTOS, S.M. - *Programação Concorrente: Mecanismos de Sincronização e Comunicação de Processos*. IV Escola de Computação, Instituto de Matemática e Estatística, USP, São Paulo, jul. 1984. 178 p.

147 - SCHROEDER, M.D. - "Engineering a Security Kernel for Multics". In: *Proc. of 5th Symposium on Operating Systems Principles*, Austin, Texas, ACM, nov. 1975. p. 25-32.

148 - SCHROEDER, M.D.; CLARK, D.D. & SALTZER, J.H. - "The Multics Kernel Design Project". In: *Proc. of 6th Symposium on Operating Systems Principles*. West Lafayette, ACM, nov. 1977. p. 43-56.

149 - SEEGMULLER, G. - "Language Aspects in Operating Systems". In: BAUER, F.L. & SAMELSON, K. (eds.) - *Language Hierarchies and Interfaces*. Berlin, Springer-Verlag, 1976. p. 266-292. (Lecture Notes in Computer Science, 46).

150 - SEGRE, L.M. & KIRNER, C. - *Primitivas de Comunicação e Sincronização de Processos por Troca de Mensagens: Uma Análise*. Relatório Técnico, Departamento de Computação e Estatística, Univ. Fed. de São Carlos, São Carlos, SP, dez. 1982. 41 p.

151 - SEIDEL, H.A. & GREBE, G. - "A Kernel for Concurrent Pascal Operating System". In: LANCIAUX, D. (ed.) - *Operating Systems: Theory and Practice*. North-Holland Publishing Company, 1979. p. 333-344.

152 - SHATZ, S.M. - "Communication Mechanisms for Programming Distributed Systems". *Computer*, 17(6):21-28, jun. 1984.

153 - SHAW, A. et alii - "A Multiprogramming Nucleus with Dinamic Resource Facilities". *Software - Practice and Experience*, 5(3):245-267, jul./sep. 1975.

154 - SHOJA, G.C. et alii - "A Control Kernel to Support Ada Interstak Communication on a Distributed Multiprocessor Computer System". *Software & Microsystems*, 1(5):128-134, aug. 1982.

155 - SHRIVASTAVA, S.K. - "Structuring Distributed Systems for Recoverability and Crash Resistence". *IEEE Transaction on Software Engineering*, SE-7(4):436-447, jul. 1981.

156 - SHRIVASTAVA, S.K. - *On the Treatment of Orphans in a Distributed System*. Technical Report Series, 188, Computing Laboratory, University of Newcastle Upon Tyne, England, aug. 1983. 8 p.

157 - SHRIVASTAVA, S.K. (ed.) - *Reliable Computer Systems*. Berlin, Springer-Verlag, 1985. 580 p.

158 - SHRIVASTAVA, S.K. & PANZIERI, F. - *The Design of a Reliable Remote Procedure Call Mechanism*. Technical Report, 171, Computing Laboratory, University of Newcastle Upon Tyne, England, oct. 1981. 18 p.

159 - SIEWIOREK, D.P. - "Architecture of Fault-Tolerance Computers". *Computer*, 17(8):9-18, aug. 1984.

160 - SILBERSCHATZ, A. - "Port Directed Communication". *The Computer Journal*, 24(1):78-82, fev. 1981.

161 - SILVA, J.L. - *Análise e Projeto de um Subsistema de Comunicação para uma Rede de Computadores*. Tese de Mestrado, Instituto de Ciências Matemáticas de São Carlos, USP, São Carlos, SP, 1986. 110 p.

162 - SILVERMAN, J.M. - "Reflections on the Verification of the Security of an Operating System Kernel". In: *Proc. of 9th Symposium on Operating Systems Principles*. Bretton Woods, New Hampshire, ACM, oct. 1983. p. 143-154.

163 - SINCOSKIE, W.D. & FARBER, D.J. - "SODS/OS: A Distributed Operating System for the IBM Series/1". *ACM Operating Systems Review*, 14(3):46-54, jul. 1980.

164 - SLOMAN, M. et alii - "A Flexible Communication System for Distributed Computer Control". In: *Proc. 5th IFAC Workshop on Distributed Computer Control Systems*. Pergamon Press, may 1983. 22 p.

165 - SLOMAN, M.; KRAMER, J. & MAGEE, J. - *Project Conic: An Architecture for Distributed Computer Control Systems*. Technical Report, Dept. of Computing, Imperial College, London, feb. 1982. 7 p.

166 - SLOMAN, M.; MAGEE, J. & KRAMER, J. - "Building Flexible Distributed Systems in CONIC". In: *Proc. SERC Distributed Computing 84 Conf.*, Bringhton, Univ. of Sussex, sep. 1984. 21 p.

167 - SOI, I.M.& AGGARWAL, K.K. - "A Review of Computer-Communication Network Classification Schemes". *IEEE Communications Magazine*, 19(2):24-32, mar. 1981.

168 - SOLOMON, M.H. & FINKEL, R.A. - "The Roscoe Distributed Operating System". In: *Proc. 7th Symposium on Operating Systems Principles*, Pacific Grove, N.Y., ACM, dec. 1979. p. 108-114.

169 - SPECTOR, A.Z. - "Performing Remote Operations Efficiently on a Local Computer Network". *Communications of the ACM*, 25(4):246-260, apr. 1982.

170 - SPECTOR, A.Z. - *Multiprocessing Architectures for Local Computer Networks*. Ph.D. Thesis, Dept. of Computer Science, Stanford University, Stanford, aug. 1981. 116 p. (Technical Report, STAN-CS-81-874).

171 - STALLINGS, W. - "Local Networks". *ACM Computing Surveys*, 16(1):3-41, mar. 1984.

172 - STALLINGS, W. - *Data and Computer Communications*. New York, N.Y., McMillan Publishing Company, 1985. 594 p.

173 - STAUNSTRUP, J. - "Message Passing Communication Versus Procedure Call Communication". *Software - Practice and Experience*, 12(3):223-234, mar. 1982.

174 - STEMPLE, D.; VINTER, S. & RAMAMRITHAM, K. - *Preliminary Design of a Port-Oriented Operating System*. Technical Report, 24, Dep. of Computer and Information Science, University of Massachusetts at Amherst, Mass., 1982. 28 p.

175 - STOTTS JR., P.D. - "A Comparative Survey of Concurrent Programming Languages". *ACM Sigplan Notices*, 17(9):76-87, sep. 82.

176 - STURGIS, H.; MITCHEL, J. & ISRAEL, J. - "Issues in the Design and Use of a Distributed File System". *ACM Operating Systems Review*, 14(3):55-69, jul. 1980.

177 - SWAN, R.J.; FULLER, S.H. & SIEWIOREK, D.P. - "Cm* - A Modular, Multi-microprocessor". In: *Proc. of National Computer Conference 77*. 1977. p. 637-644.

178 - TANENBAUM, A.S. - *Computer Networks*. Englewood Cliffs, N.J., Prentice-Hall, Inc., 1981. 517 p.

179 - TANENBAUM, A.S. & MULLENDER, S.J. - "An Overview of the Amoeba Distributed Operating Systems". *ACM Operating Systems Review*, 15(3):51-64, jul. 1981.

180 - TEIXEIRA, C.A.C. - "Suporte para Desenvolvimento de Sistemas Distribuídos: Arquitetura do Processador de Comunicação". In: *Anais do 4º Congresso da Sociedade Brasileira de Computação - 11º SEMISH*. Viçosa, MG, SBC, jul. 1984. p. 465-468.

181 - TEIXEIRA, C.A.C. & TREVELIN, L.C. - "Protocolo de Acesso e Superação de Falhas em Redes com Topologia Estrela Coincidente". In: *Anais do 2º Simpósio sobre Redes de Computadores*. Campina Grande, Paraíba, UFPb, abr. 1984. p. 16.1-16.15.

182 - THEAKER, C.J. & FRANK, G.R. - "MUSS - A Portable Operating System". *Software - Practice and Experience*, 9(8):633-643, aug. 1979.

183 - THEAKER, C.J. & FRANK, G.R. - "An Assessment of the MUSS Operating System". *Software - Practice and Experience*, 9(8):657-670, aug. 1979.

184 - THORELLI, L.E. - "A Monitor for Small Computers". *Software - Practice and Experience*, 8(4):439-450, jul./aug. 1978.

185 - THURBER, K.J. & MASSON, G.M. - *Distributed - Processor Communication Architecture*. Lexington, Mass., D.C. Heath and Company, 1979. 253 p.

186 - TISATO, F. & ZICARI, R. - *Operating System and System Language Requirement List*. Public. Int. del Progetto Finalizzato Informatica C.N.R. - Progetto Cnet, 19, Pisa, Itália, dez. 1980. 19 p.

187 - TOKORO, M. et alii - "A High Level Multi-Lingual Multiprocessor KMP/II". In: *Proc. of the 7th Annual Symposium on Computer Architecture*. La Baule, France, ACM, may 1980. p. 325-333.

188 - TOKUDA, H. & MANNING - "An Interprocess Communication Model for a Distributed Software Testbed". In: *Proc. of SIGCOMM'83 Symposium on Communications Architectures & Protocols*. Austin, Texas, ACM, mar. 1983. p. 205-212.

189 - TSAY, D.P. & LIU, M.T. - "MIKE: A Network Operating System for the Distributed Double-Loop Computer Network". *IEEE Transaction on Software Engineering*, SE-9(2):143-154, mar. 1983.

190 - VAN TILBORG, A.M. & WITTIE, L.D. - "Operating Systems for the Micronet Network Computer". *IEEE Micro*, 3(2):38-47, apr. 1983.

191 - VINTER, S.; RAMAMRITHAM, K. & STEMPLE, D. - *Protecting Objects Through the Use of Ports*. Technical Report, 23, Dep. of Computer and Information Science, University of Massachusetts at Amherst, Mass., 1982. 22 p.

192 - VON ISSENDORFF, H. & GRÜNEWALD, W. - "An Adaptable Network for Functional Distributed Systems". In: *Proc. of the 7th Annual Symposium on Computer Architecture*. La Baule, France, ACM, may 1980, p. 196-201.

193 - WALKER, B.J. et alii - "The LOCUS Distributed Operating System". In: *Proc. of 9th Symposium on Operating Systems Principles*. Bretton Woods, New Hampshire, ACM, oct. 1983. p. 49-70.

194 - WALKER, B.J.; KEMMERER, R.A. & POPEK, G.J. - "Specification and Verification of UCLA Unix Security Kernel". *Communication of the ACM*, 23(2):118-131, feb. 1980.

195 - WALTER, K.G. et alii - "Structured Specification of a Security Kernel". In: *Proc. of Int. Conf. Reliable Software*, Los Angeles, Califórnia, apr. 1975. p. 285-293.

196 - WATSON, R.W. & FLETCHER, J.G. - "An Architecture for Support of Network Operating System Services". *Computer Networks*, 4(1):33-49, feb. 1980.

197 - WEGNER, P. - "The ADA Language and Environment". *ACM SIGSOFT Engineering Notes*, 5(2):8-14, apr. 1980.

198 - WEITZMAN, C. - *Distributed Micro/Minicomputer Systems: Structure, Implementation, and Application*. Englewood Cliffs, N.J., Prentice-Hall, Inc., 1980. 403 p.

199 - WILLIAMSON, R. & HOROWITZ, E. - "Concurrent Communication and Synchronization Mechanisms". *Software - Practice and Experience*, 14(2):135-151, feb. 1984.

200 - WIRTH, N. - "Modula: A Language for Modular Multiprogramming". *Software - Practice and Experience*, 7(1): 3-35, jan./feb. 1977.

201 - WIRTH, N. - *Modula-2*. Technical Report, Institut für Informatik, ETH, Zürich, mar. 1980. 36 p.

202 - WIRTH, N. - *Programming in Modula-2*. Berlin, Springer-Verlag, 1982. 176 p.

203 - WITTIE, L.D. & VAN TILBORG, A.M. - "MICROS, A Distributed Operating System for MICRONET, A Reconfigurable Network Computer". *IEEE Transaction on Computers*, C-29(12): 1133-1144, dec. 1980.

204 - WULF, W. et alii - "HYDRA: The Kernel of a Multiprocessor Operating System". *Communications of the ACM*, 17(6): 337-345, jun. 1974.

205 - ZHONGXIU, S.; DU, Z. & PEIGEN, Y. - "ZCZOS: A Distributed Operating System for a LSI-11 Microcomputer Network". *ACM Operating System Review*, 17(3):30-34, jul. 1983.

206 - ZWAENEPOEL, W. & LANTZ, K.A. - "Perseus: Retrospective on a Portable Operating System". *Software - Practice and Experience*, 14(1):31-48, jan. 1984.

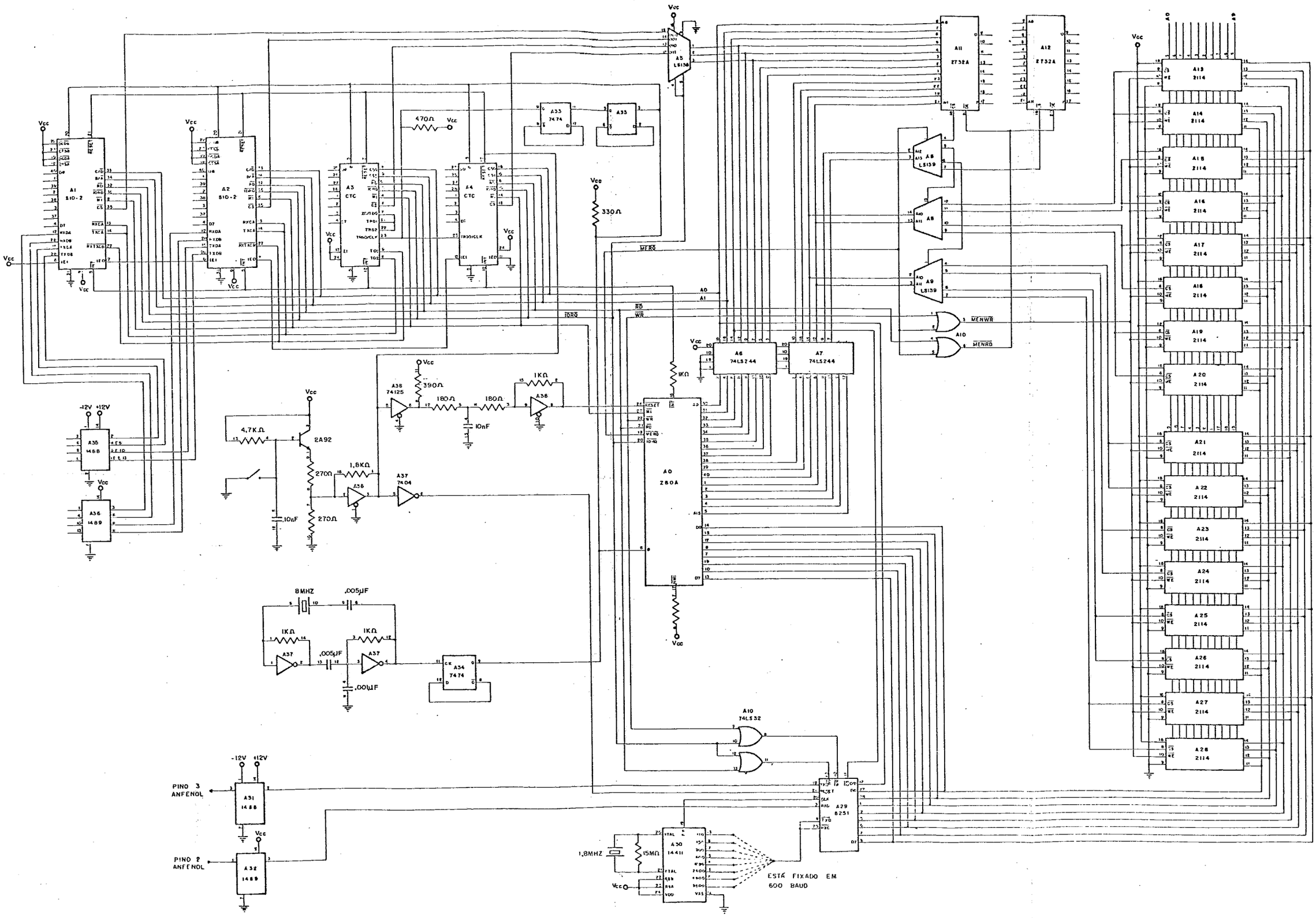


FIGURA A.1 - CIRCUITO DO PROCESSADOR DE COMUNICAÇÃO PARA CONEXÃO PONTO A PONTO

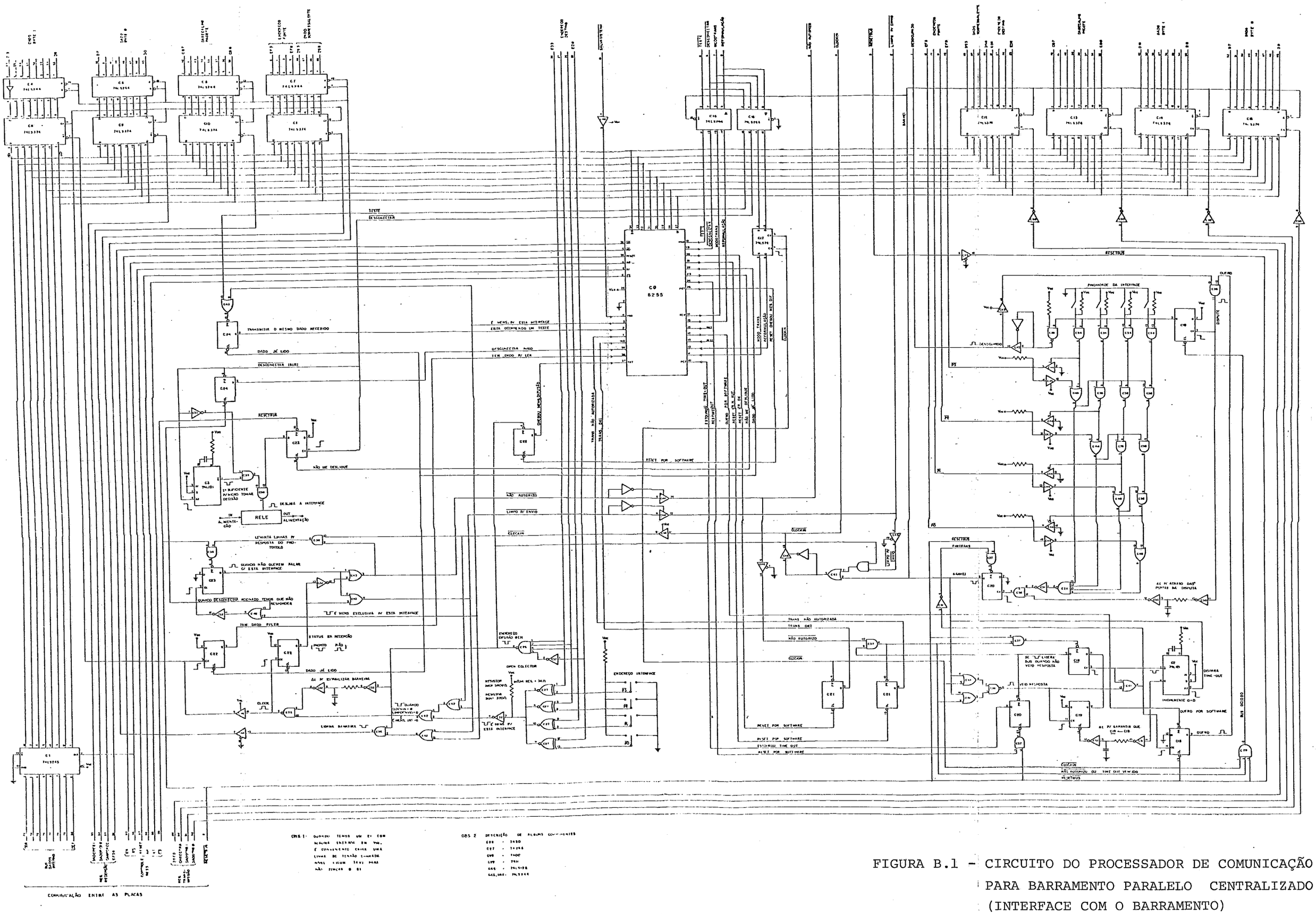


FIGURA B.1 - CIRCUITO DO PROCESSADOR DE COMUNICAÇÃO PARA BARRAMENTO PARALELO CENTRALIZADO (INTERFACE COM O BARRAMENTO)

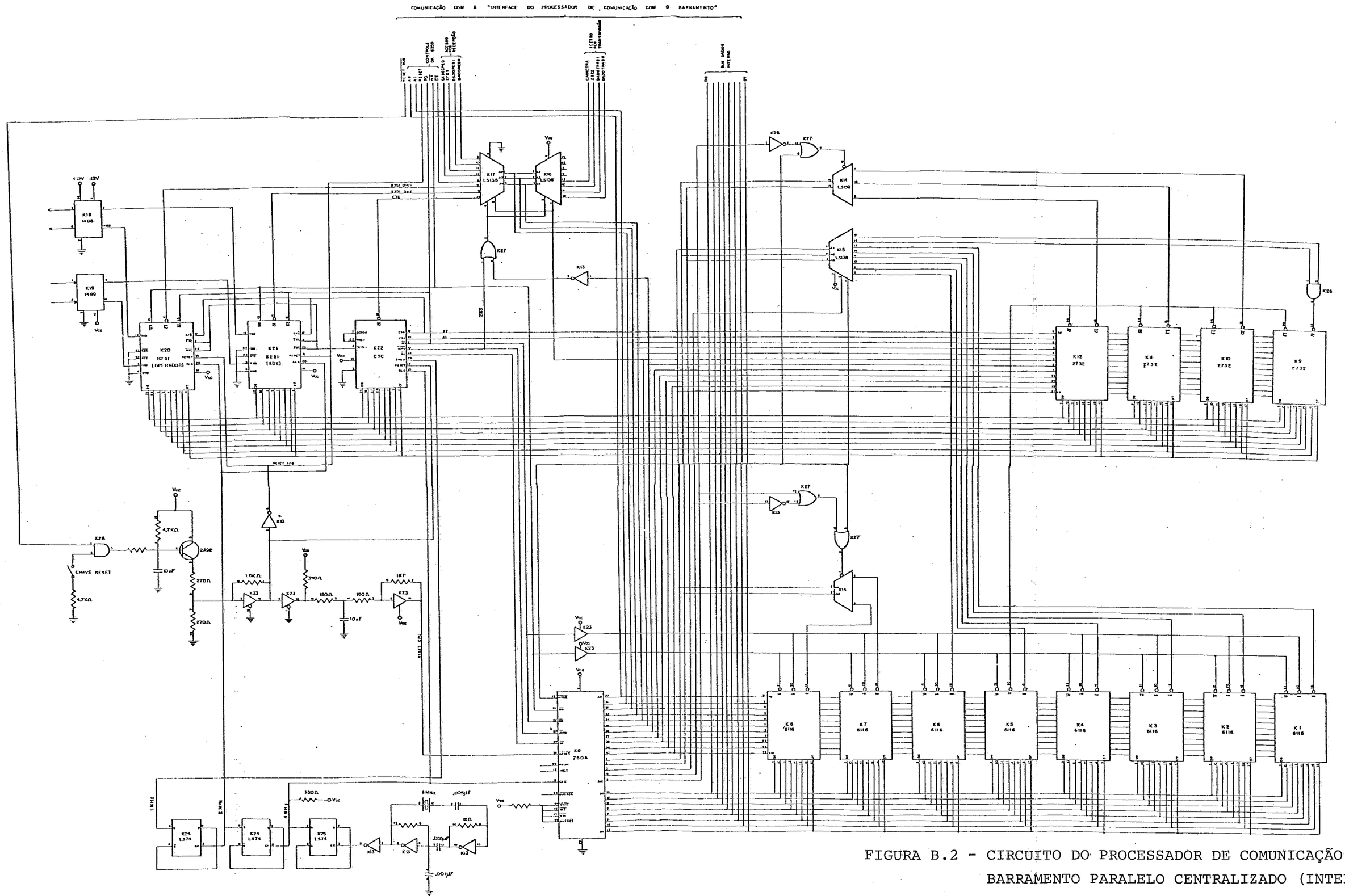


FIGURA B.2 - CIRCUITO DO PROCESSADOR DE COMUNICAÇÃO PARA BARRAMENTO PARALELO CENTRALIZADO (INTERFACE COM O PROCESSADOR OPERADOR E CONTROLE DAS INTERFACES)

APÊNDICE C

ALGORÍTMO DE ROTEAMENTO

```
procedimento roteamento (nó origem, nó destino, var sentido, var
                         possível);
   ⋮
início
se a tabela de conexões indicar que não há falha na conexão destino
   então
      se a tabela de laços indicar que há alguma falha em laço
         então
            ┌ ESTABELECIMENTO DA ROTA COM FALHA EM LAÇO
            └ (* ver algorítmo C.2 *)
         senão
            ┌ ESTABELECIMENTO DA ROTA SEM FALHA
            └ (* ver algorítmo C.3 *)
   senão
      possível ← falso
fim;
```

ALGORÍTMO C.1 - DETERMINAÇÃO DA ROTA

(continuação do Apêndice C)

```
início
descobrir o primeiro laço com falha no sentido horário e anti-horário;
calcular a distância do nó origem ao laço com falha no sentido horário;
calcular a distância do nó origem ao nó destino no sentido horário;
se a distância ao laço com falha no sentido horário for menor ou
   igual à distância ao nó destino no sentido horário
   então
      início
      calcular a distância do nó origem ao laço com falha no sentido anti-horário;
      calcular a distância do nó origem ao nó destino no sentido
      anti-horário;
      se a distância ao laço com falha no sentido anti-horário
         for menor ou igual à distância ao nó destino no sentido
         horário
         então possível ← falso;
         senão
            início
            sentido ← anti-horário
            possível ← verdadeiro
            fim
      fim
   senão
      início
      sentido ← horário;
      possível ← verdadeiro
      fim
fim
```

ALGORÍTMO C.2 - ESTABELECIMENTO DA ROTA COM FALHA EM LAÇO

(continuação do Apêndice C)

início
possível ← *verdadeiro*
calcular a distância do nó origem ao nó destino no sentido horário;
calcular a distância do nó origem ao nó destino no sentido anti-horário;
se a distância ao nó destino no sentido horário for maior que a distância ao nó destino no sentido anti-horário
então sentido ← anti-horário
senão sentido ← horário
fim

ALGORÍTMO C.3 - ESTABELECIMENTO DA ROTA SEM FALHA

A P E N D I C E D

PROGRAMA SUPERVISOR DO PROCESSADOR DE COMUNICAÇÃO

```
(* supervisor localizado no nó origem *)
início
período de manutenção de falhas da tabela ← 0;
marca de transmissão ← posição inicial;
enquanto processador estiver ligado
   faça
   início

   ┌ C I C L O  DE RECEPÇÃO  ┐
   └ (* ver algorítmo D.2 *) ┘

   ┌ C I C L O  DE TRANSMISSÃO ┐
   └ (* ver algorítmo D.3 *)   ┘

   (* fase de eliminação de falhas da tabela *)

   se período de manutenção de falhas da tabela atingiu o limite
      então
         início
         retirar falhas da tabela;
         período de manutenção de falhas da tabela ← 0
         fim
      senão incrementar período de manutenção de falhas da  tabe
                  la
   fim
fim
```

ALGORÍTMO D.1 - SUPERVISOR DO PROCESSADOR DE COMUNICAÇÃO

(continuação do Apêndice D)

```
(* ciclo de recepção de até 3 pacotes *)
execute para "buffer" de recepção variando de 1 até 3
   início
   tentar transferir o pacote do "buffer" de recepção corrente
   para o "buffer" de trabalho;
   se houve sucesso na transferência
      então
         se o destino do pacote for igual ao nó origem
            então
               se o pacote indicar operação do processador de co
                  municação
                  então executar a operação
                  senão transferir o pacote do "buffer" de traba
                        lho para a fila de pacotes ligada à cone
                        xão com o processador operador
            senão transferir o pacote do "buffer" de trabalho pa
                  ra uma fila de pacotes ligada a um laço, usan
                  do o roteamento quando o pacote estiver vindo
                  do processador operador local
   fim
```

ALGORÍTMO D.2 - CICLO DE RECEPÇÃO

(continuação do Apêndice D)

```
(* ciclo de transmissão de 1 pacote *)
avançar ponteiro de transmissão a partir da marca de transmissão;
varredura ← 1;
sucesso na transmissão ← falso;
enquanto não houver sucesso na transmissão e não completar a var
        redura das 3 filas de pacotes (pares)
   faça
   início
   se a fila de pacotes indicada pelo ponteiro de transmissão não
      estiver vazia
      então
         início
         tentar transmitir um pacote da fila corrente;
         se houve sucesso na transmissão
            então
               início
               marca de transmissão ← ponteiro de transmissão;
               sucesso na transmissão ← verdadeiro
               fim
            senão
               se houve falha na transmissão
                  então
                     início
                     tentar transmitir com falha;
                     se houve sucesso na transmissão com falha
                        então
                           início
                           marca de transmissão ← ponteiro de
                                                  transmissão;
                           sucesso na transmissão ← verdadeiro
                           fim
                        senão avançar ponteiro de transmissão
                     fim
                  senão avançar ponteiro de transmissão
         fim
      senão avançar ponteiro de transmissão;
   se sucesso na transmissão for igual a falso
      então incrementa a varredura
   fim
```

ALGORÍTMO D.3 - CICLO DE TRANSMISSÃO

APÊNDICE E

ESTRUTURAÇÃO DO PROGRAMA SUPERVISOR EM NÍVEIS DE IMPLEMENTAÇÃO

PROCEDIMENTOS DE NÍVEL-1 (terminais)

a) *Testar Operação*
   - Se o pacote alcançou seu destino, verificar se ele está trazendo alguma requisição de operação a ser executada pelo processador de comunicação.

b) *Procurar Primeira Mensagem*
   - Varre a fila de pacotes obtida pela concatenação da fila de pacotes curtos com a fila de pacotes longos, e localiza o primeiro pacote longo (mensagem).

c) *Preparar a Substituição do Anúncio em Atendimento*
   - Gera as condições para que o anúncio em atendimento seja substituído por outro da fila de anúncios.

d) *Copiar Elemento-Fila*
   - Faz um "buffer" receber um pacote de uma fila ou vice-versa.

e) *Desligar Índice de Fila*
   - Retira um elemento da fila de índices que apontam estruturas de dados.

f) *Ligar Índice na Fila*
   - Coloca um elemento na fila de índices que apontam estruturas de dados.

g) *Transmitir Buffer*
   - Tenta transmitir o "buffer" de trabalho para um "buffer" de recepção vizinho.

h) *Ler Buffer*

- Tenta transferir um "buffer" de recepção local para o "buffer" de trabalho.

i) *Testar Laço*

- Verifica se um laço específico está com falha (usado na simulação).

j) *Testar Conexão*

- Verifica se uma conexão específica está com falha (usado na simulação).

k) *Ler Status*

- Lê o valor do "status" de um "buffer" de recepção, indicando se contém ou não informação.

ℓ) *Atualizar Status*

- Atribui um valor ao "status" de um "buffer" de recepção, indicando se contém ou não informação (usado também na simulação).

m) *Retirar Falhas da Tabela*

- A partir de um ponto aleatório, varre as tabelas indicativas de falha nos laços e nas conexões e retira a primeira anotação que encontrar.

n) *Atualizar Variável de Falha no Laço*

- Faz a varredura na tabela indicativa de falha nos laços e faz a variável indicativa de alguma falha ficar *verdadeira*, caso haja falha, ou *falsa*, caso contrário.

o) *Descobrir Laço com Falha*

- Faz uma varredura na tabela de laços e anota o primeiro laço com falha no sentido horário e no sentido anti-horário.

PROCEDIMENTOS DE NÍVEL-2

a) *Testar Falha na Ligação*

- Verifica se a ligação (conexão ou laço) está com falha (usado na simulação).

- Procedimentos usados:
  . Testar Laço (nível-1);
  . Testar Conexão (nível-1).

b) *Roteamento*
- Determina a fila de pacotes para onde irá um pacote recebido.
- Procedimentos usados:
  . Descobrir Falha no Laço (1).

c) *Inserir Elemento na Fila*
- Coloca um "buffer" em uma fila de pacotes ou de anúncio.
- Procedimentos usados:
  . Desligar Índice de Fila (1);
  . Ligar Índice na Fila (1);
  . Copiar Elemento-Fila (1).

d) *Transferir Índice de Fila*
- Muda o ponteiro de uma estrutura de dados de uma fila para outra.
- Procedimentos usados:
  . Desligar Índice da Fila (1);
  . Ligar Índice na Fila (1).

PROCEDIMENTOS DE NÍVEL-3

a) *Varrer e Destruir/Colocar Anúncio na Fila*
- Procura um anúncio na fila para destruí-lo ou verifica se um anúncio já está na fila antes de colocá-lo.
- Procedimentos usados:
  . Transferir Índice de Fila (2);
  . Copiar Elemento-Fila (2).

b) *Retirar Elemento da Fila*
- Faz um elemento ("buffer") receber um pacote da fila.
- Procedimentos usados:
  . Transferir Índice de Fila (2);
  . Copiar Elemento-Fila (2).

c) *Inserir Pacote Curto*

- Procura o ponto de junção na concatenação da fila de pacotes curtos com a fila de pacotes longos, e insere o pacote curto nesse ponto (prioridade para pacotes curtos).
- Procedimentos usados:
  . Procurar Primeira Mensagem (1);
  . Inserir Elemento na Fila (2).

PROCEDIMENTOS DE NÍVEL-4

a) *Testar e Inserir Elemento de Fila*

- Testa as condições da concatenação das filas de pacotes curtos e longos, e insere um elemento de uma fila, colocado no "buffer" de trabalho, em outra fila.
- Procedimentos usados:
  . Inserir Pacote Curto (3);
  . Transferir Índice de Fila (2);
  . Inserir Elemento na Fila (2).

b) *Testar e Destruir Anúncio na Fila*

- Procura anúncio na fila e o destrói.
- Procedimentos usados:
  . Varrer e Destruir/Colocar Anúncio na Fila (3).

c) *Testar, Inserir e Atualizar Elemento de Buffer*

- Testa as condições da cancatenação das filas de pacotes curtos e longos, e insere um elemento de um "buffer" de recepção colocado no "buffer" de trabalho, atualizando o "status" do "buffer" de recepção.
- Procedimentos usados:
  . Inserir Pacote Curto (3);
  . Atualizar "Status" (1);
  . Inserir Elemento na Fila (2).

d) *Testar e Colocar Anúncio na Fila*

- Testa as condições da fila de anúncios e coloca o anúncio na fila.

- Procedimentos usados:
  . Varrer e Destruir/Colocar Anúncio na Fila (3).

e) *Avisar Falha*
- Monta os pacotes de aviso de falha e coloca-os numa fila de pacotes ligada a um laço.
- Procedimentos usados:
  . Inserir Pacote Curto (3).

f) *Transmitir Elemento de Fila*
- Se as condições de transmissão por uma conexão ou laço for favorável, retirar um pacote de uma fila, colocando-o no "buffer" de trabalho e transferindo-o para o "buffer" de recepção destino. Atualiza o "status" do "buffer" de recepção destino.
- Procedimentos usados:
  . Ler Status (1);
  . Retirar Elemento da Fila (3);
  . Transmitir Buffer (1);
  . Atualizar Status (3).

PROCEDIMENTOS DE NÍVEL-5

a) *Devolver Resposta*
- Monta um pacote de controle, em resposta a uma requisição de operação, e o devolve ao emissor da requisição.
- Procedimentos usados:
  . Roteamento (2);
  . Testar, Inserir e Atualizar Elemento do Buffer (4).

b) *Transferir Buffer*
- Transfere o "buffer" de trabalho para uma fila de pacotes e atualiza o "status" do "buffer" de recepção de onde foi lido o pacote.
- Procedimentos usados:
  . Testar, Inserir e Atualizar Elemento de Buffer (4);
  . Roteamento (2).

c) *Transmitir com Falha*

- Verifica os campos de falha do pacote, procurando anotar a falha. Em seguida, reverte o trajeto do pacote ou devolve-o à sua origem, dependendo das circunstâncias. Aproveita para comunicar a existência da falha aos outros nós.

- Procedimentos usados:
  . Roteamento (2);
  . Testar, Inserir e Atualizar Elemento do Buffer (4);
  . Transferir Índice de Fila (2);
  . Avisar Falha (4).

PROCEDIMENTOS DE NÍVEL-6

a) *Executar Operação*

- Executa operação a nível do processador de comunicação, atendendo uma requisição externa, ou cuidando de questões relacionadas com falhas.

- Procedimentos usados:
  . Testar e Destruir Anúncio na Fila (4);
  . Preparar a Substituição do Anúncio em Atendimento (1);
  . Testar e Colocar Anúncio na Fila (4);
  . Devolver Resposta (5);
  . Retirar Elemento da Fila (3);
  . Atualizar "Status" (1).

PROCEDIMENTOS DE NÍVEL-7

a) *Supervisor do Processador de Comunicação*

- Gerencia o processador de comunicação, de forma a controlar o fluxo de informação na rede e a comunicação com o processador operador, além de cuidar das questões relacionadas com falhas.

- Procedimentos usados:
  . Ler Status (1);
  . Testar Operação (1);
  . Executar Operação (6);
  . Transferir Buffer (5);

. Testar Falha na Ligação (2);
. Transmitir Elemento de Fila (4);
. Transmitir com Falha (5).

APÊNDICE F

CONSIDERAÇÕES SOBRE A SIMULAÇÃO DA REDE EM LAÇO COM CONEXÃO PONTO A PONTO

Os processadores de comunicação e processadores operadores da rede apresentam um funcionamento concorrente por natureza, de forma que seria normal a utilização de linguagens concorrentes como Pascal Concorrente, CSP/K, Módula-2, etc., para a simulação de seu comportamento. Como não se tinha disponível nenhuma dessas linguagens, desenvolveu-se uma estrutura de concorrência que permitisse a utilização da linguagem Pascal (sequencial).

A estruturação do ambiente consistiu em transformar-se cada programa supervisor do processador de comunicação e cada programa aplicativo do processador operador, incluindo as chamadas das primitivas, em trechos cíclicos do programa principal, correspondendo aos processos concorrentes. Para poder obter a concorrência, introduziu-se, nesses trechos de programa, desvios forçados para um outro trecho de programa denominado escalador de processos. O salvamento e restauração do contexto dos processos nos vários pontos será feito com base em rótulos. Quando ocorrer um desvio forçado para o escalador, um rótulo será colocado na fila de processos do escalador, e outro rótulo será retirado desta fila para indicar o ponto de continuação da execução.

Para introduzir a concorrência não determinística, colocou-se um teste de variável aleatória antes do desvio forçado, permitindo ou o desvio forçado, ou a continuação do mesmo processo até o próximo ponto de desvio, dependendo do valor aleatório. Um valor aleatório também poderá influenciar na escolha de um elemento da fila de escalação para continuar a execução.

A fila de processos do escalador conterá, no começo, os pontos iniciais de entrada dos vários processos. Quando um processo escapar do ciclo para terminar, ele preparará seus parâmetros de controle de forma que o escalador não faça a inserção

de um rótulo. Desta maneira, quando todos os processos terminarem, a fila de processos do escalador ficará vazia, e o simulador terminará a sua execução.

Apresenta-se a seguir a estruturação simplificada da simulação da rede, consistindo dos algorítmos relativos a: programa de simulação da rede em laço; detalhe do trecho de programa do escalador; e detalhe do trecho de programa do supervisor do processador de comunicação-1. As estruturas dos outros trechos de programa são parecidas com a do trecho de programa do supervisor de comunicação-1.

(continuação do Apêndice F)

```
    ...
    procedimentos do supervisor do processador de comunicação;
    procedimentos das primitivas de comunicação;
    outros procedimentos;
    início
    estabelecimento das condições iniciais;
 1: [ TRECHO DE PROGRAMA DO ESCALADOR
    (* ver algorítmo F.2 *) ]
10: [ TRECHO DE PROGRAMA DO SUPERVISOR DO PROCESSADOR DE
    COMUNICAÇÃO-1
    (* ver algorítmo F.3 *) ]
20: [ TRECHO DE PROGRAMA DO SUPERVISOR DO PROCESSADOR DE
    COMUNICAÇÃO-2 ]
    :
50: [ TRECHO DE PROGRAMA DO SUPERVISOR DO PROCESSADOR DE
    COMUNICAÇÃO-5 ]
60: [ TRECHO DE PROGRAMA DO PROCESSADOR OPERADOR-1 ]
    :
100: [ TRECHO DE PROGRAMA DO PROCESSADOR OPERADOR-5 ]
110: [ TRECHO DE PROGRAMA DO GERADOR ALEATÓRIO DE FALHAS ]
999: fim
```

ALGORÍTMO F.1 - PROGRAMA DE SIMULAÇÃO DA REDE EM LAÇO

(continuação do Apêndice F)

```
   (* escalador de processo *)
1: se chamada normal
      então
         colocar rótulo na fila;
   se a fila estiver vazia
      então vá para 999
      senão
         início
         retirar rótulo da fila;
         execute o código do rótulo igual a
            10: vá para 10;
            11: vá para 11;
            12: vá para 12;
            :
            20: vá para 20;
            21: vá para 21;
            :
            119: vá para 119
            fim
         fim
```

ALGORÍTMO F.2 - DETALHE DO TRECHO DE PROGRAMA DO ESCALADOR

(continuação do Apêndice F)

```
    (* processador de comunicação-1 *)
10: início
    estabelecimento das condições iniciais;
    enquanto ...
        faça
        início
        :
        se aleatório(N) > N/2
           então
              início
              chamada ← normal;
              rótulo  ← 11; (* ponto de continuação do processo *)
              vá para 1 (* desvio para o escalador *)
              fim;
    11: procedimento ...
        :
        se aleatório(N) > N/2
           então
              início
              chamada ← normal;
              rótulo  ← 17; (* ponto de continuação do processo *)
              vá para 1 (* desvio para o escalador *)
              fim;
    17: procedimento ...
        :
        fim
        chamada ← anormal; (* para não inserir rótulo *)
        vá para 1 (* desvio para o escalador *)
```

ALGORÍTMO F.3 - DETALHE DO TRECHO DE PROGRAMA DO SUPERVISOR DO PROCESSADOR DE COMUNICAÇÃO-1