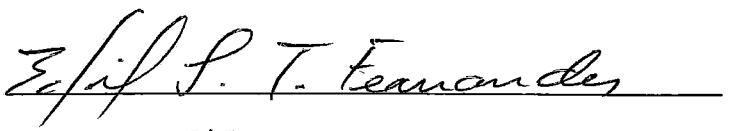
UM NÚCLEO DE SISTEMA OPERACIONAL DISTRIBUÍDO

PARA A MÁQUINA PARALELA HÍBRIDA

Myrian Christina de Aragão Marques

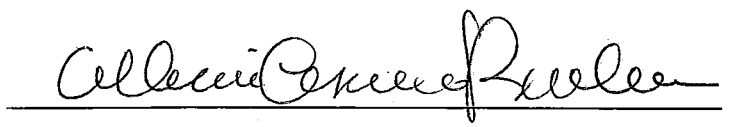
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO

Aprovada por:

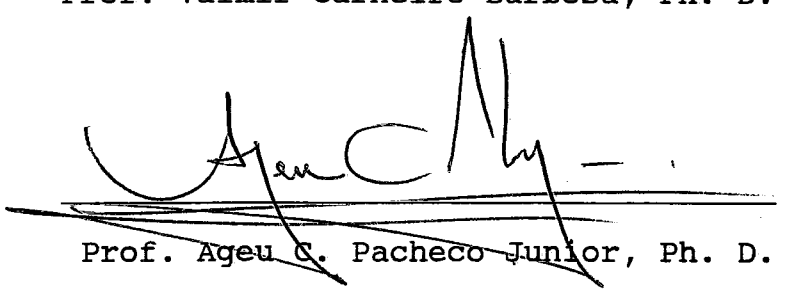


Prof. Edil S. T. Fernandes, Ph. D.

(Presidente)



Prof. Valmir Carneiro Barbosa, Ph. D.



Prof. Ageu C. Pacheco Junior, Ph. D.

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 1990

MARQUES, MYRIAN CHRISTINA DE ARAGÃO

Um Núcleo de Sistema Operacional Distribuído para a Máquina Paralela Híbrida [Rio de Janeiro] 1990

XIV, 133 p. 29,7 cm (COPPE/UFRJ, M. Sc., Engenharia de Sistemas e Computação, 1990)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas Distribuídos I. COPPE/UFRJ

II.Título(série)

Esta tese é dedicada:

- ao meu marido Silvio

AGRADECIMENTOS

Aos professores Edil Severiano Tavares Fernandes e Valmir Carneiro Barbosa por toda a paciência e presteza durante a orientação desta tese.

Aos Engenheiros, Técnicos, Estagiários e Bolsistas do Laboratório de Sistemas e Computação da COPPE e principalmente à Alberto F. Souza e José D. Queiroz , pela inestimável ajuda na elaboração e testes deste trabalho.

À amiga Ana Dolejsi dos Santos pelo incentivo e orientação para um melhor desenvolvimento deste trabalho.

Aos amigos da Petrobrás, Alexandre Garcia, Vítor Lisboa, Fernando Maurício e Ricardo Padilha pela ajuda direta e até mesmo indireta durante os meses de minha dedicação a este trabalho.

Aos amigos Darlene, Roberto, Gilberto, Pedro, Anis e especialmente à Márcia e Daise pela força e valioso auxílio na elaboração das figuras.

À minha amiga Vanise pelo apoio e preocupação em alguns momentos muito difíceis.

À minha família por toda a compreensão durante esses meses.

Um agradecimento especial ao meu marido Silvio, que

além do auxílio na edição e elaboração desta tese, teve todo o carinho e força suficiente para me transmitir a calma e o amor que precisei.

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

UM NÚCLEO DE SISTEMA OPERACIONAL DISTRIBUÍDO PARA A MÁQUINA PARALELA HÍBRIDA

Myrian Christina de Aragão Marques

ABRIL , 1990

Orientadores : Edil Severiano Tavares Fernandes
Valmir Carneiro Barbosa

Programa : Engenharia de Sistemas e Computação

Este trabalho descreve a especificação e implementação do Núcleo de um Sistema Operacional para a MPH - Máquina Paralela Híbrida. A MPH foi desenvolvida pelo Grupo de Arquitetura de Computadores e Sistemas Operacionais da COPPE/UFRJ em 1987. Esta máquina consiste de diversos nós de processamento, organizados segundo um hipercubo de dimensão n. Os processadores operam em paralelo e empregam um mecanismo de comunicação baseado em troca de mensagens.

Utilizando conceitos de Sistemas Operacionais Distribuídos desenvolvemos um sub-sistema de troca de mensagens, rotinas para gerenciamento do meio-ambiente concorrente, e as funções providas ao usuário.

No primeiro capítulo deste trabalho introduz-se os

conceitos básicos de Sistemas Operacionais para máquinas do tipo multiprocessador. No segundo capítulo descreve-se a aruitetura da hospedeira, isto é, da Máquina Paralela Híbrida. Nos dois capítulos seguintes apresenta-se respectivamente a especificação e a implementação do Núcleo. Finalmente, no último capítulo relata-se as principais conclusões obtidas.

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

A KERNEL OF A DISTRIBUTED OPERATING SYSTEM FOR THE PARALLEL HYBRID MACHINE

Myrian Christina de Aragão Marques

ABRIL , 1990

Advisors : Edil Severiano Tavares Fernandes
Valmir Carneiro Barbosa
Department : Engenharia de Sistemas e Computação

This work describes the specification and implementation of the Kernel of an Operating System for the MPH - Parallel Hybrid Machine. The MPH was developed by the Computer Architecture and Operating Systems Group of the COPPE/UFRJ, in 1987. This machine consists of several processing nodes organized as an N-dimensional hypercube. The processors work in parallel and use a communication mechanism based on message passing.

Using Distributed Operating Systems concepts we developed a message passing subsystem, routines for the management of the concurrent environment, and the functions provided to the user.

The first chapter of this work presents the basic

concepts of Operating Systems for multiprocessor machines. The second chapter describes the architecture of the host (i.e., the Parallel Hybrid Machine). The third and fourth chapters present the specification and implementation of the Kernel. Finally, the last chapter relates the main conclusions.

Índice

ÍNDICE DAS FIGURAS

CAPÍTULO II

CAPÍTULO III

CAPÍTULO I

INTRODUÇÃO

A principal característica de um **Sistema Operacional Multiprocessado** é a capacidade de suportar o paralelismo tanto a nível de máquina quanto a nível de programas. O projeto de uma máquina com dois ou mais processadores apresenta um custo de "hardware" e "software" maior do que o custo de uma máquina com um processador. Os benefícios originados da colocação de processadores devem exceder os custos, explorando-se o máximo de paralelismo possível em todos os níveis do Sistema Operacional, de forma a oferecer ao usuário facilidades e conforto na utilização da máquina.

Existe uma grande diferença entre os Sistemas Operacionais de multiprocessadores e uniprocessadores em relação à estrutura e organização do sistema de processadores. Três organizações básicas têm sido utilizadas, DEITEL(1984):

- Sistema Mestre/Escravo;
- Sistema com Executivo separado em cada processador;
- Sistema com Tratamento Simétrico para todos os processadores.

Na organização Mestre/Escravo, somente um processador, o Mestre, executa o Sistema Operacional, enquanto que os outros processadores, os Escravos, executam programas de usuário. Quando um processo de usuário requisita uma função do Sistema Operacional, o Escravo gera uma interrupção e espera até que o Mestre atenda o seu pedido e execute a

função. As interrupções geradas pelos Escravos são enfileiradas para serem atendidas pelo Mestre. Esta organização, apesar de ser de fácil implementação, é pouco confiável, pois qualquer falha no Mestre pode causar uma falha catastrófica em todo o sistema.

Em um Sistema com Executivo Separado, cada processador possui o seu Sistema Operacional, manipulando as interrupções para o processador, recebendo as requisições dos processos e controlando seus recursos dedicados, como em um sistema com um único processador. Um processo executa, desde o início até o término, em um único processador. Algumas tabelas possuem informações globais como, por exemplo, os processadores conhecidos no sistema, e o acesso a essas tabelas é controlado por técnicas de exclusão mútua. Neste tipo de sistema é improvável que uma falha em um processador ocasione uma falha catastrófica.

O Sistema com Tratamento Simétrico é o de mais complexa implementação. O Sistema Operacional possui um "pool" de processadores idênticos, que podem ser utilizados em qualquer função do sistema. O Sistema Operacional flutua de um processador para outro e o processador que no momento cuida das tabelas e das funções do sistema é chamado executivo. Somente um processador pode ser o executivo de cada vez. Um processo pode ser executado em instantes diferentes por qualquer dos processadores. Por essas características, torna-se necessário o uso de código reentrante e exclusão mútua. A carga de todo o sistema pode ser balanceada mais facilmente, pois todo o trabalho pode

ser roteado para qualquer processador disponível. Quando um processador falhar, o Sistema Operacional fará sua remoção e notificará o operador.

Com a motivação para criação de um ambiente de ensino e pesquisa em processamento paralelo, iniciou-se em 1987, pelo Grupo de Arquitetura e Sistemas Operacionais do Programa de Engenharia de Sistemas e Computação da COPPE/UFRJ, o desenvolvimento da **Máquina Paralela Híbrida** e, atualmente, do Núcleo de um Sistema Operacional Distribuído de Gerenciamento desta Máquina.

Devido às características da **Máquina Paralela Híbrida**, que apesar de seu baixo custo fornece as facilidades essenciais à computação paralela, optou-se pela organização de um Sistema Operacional com um Núcleo executando separado em cada processador e todos os processos cooperando na execução de qualquer processo da Máquina.

Este trabalho documenta o desenvolvimento deste Núcleo que oferece o suporte básico à operação da **Máquina Paralela Híbrida.**

No **capítulo II**, a **Máquina Paralela Híbrida** será apresentada e suas características comentadas. Algumas soluções pertinentes ao escopo deste trabalho são analisadas, assim como o estado atual da Máquina e sugestões de aplicações.

O **capítulo III** descreve a especificação do Núcleo do

Sistema Operacional. São apresentadas as facilidades desejáveis, as funções executadas e a estrutura do Núcleo.

O **capítulo IV** contém a implementação, a descrição das estruturas utilizadas, dos algoritmos codificados e alguns detalhes como, por exemplo, certas restrições devido ao uso da linguagem ou da característica da máquina. Ainda neste capítulo são relatadas as etapas de implementação e teste do Núcleo.

O **capítulo V** apresenta uma conclusão com a apresentação dos resultados de alguns testes e sugestões para futuros aperfeiçoamentos do sistema.

O **apêndice A** contém a listagem do código fonte do Núcleo.

O **apêndice B** apresenta a demonstração do algoritmo da área de memória compartilhada.

CAPÍTULO II

A MÁQUINA PARALELA HÍBRIDA

II.1 - Introdução

A Máquina Paralela Híbrida, de agora em diante chamada MPH, possui uma arquitetura do tipo MIMD ("Multiple Instruction Stream Multiple Data Stream"), na qual diversas instruções podem ser executadas em paralelo por mais de um processador, a partir de diferentes fluxos de instruções.

Os processadores na MPH estão interligados de acordo com a topologia hipercúbica, onde cada processador é ligado a n processadores vizinhos, sendo n a ordem do hipercubo. A implementação atual da MPH se constitui de um hipercubo de ordem 2, ou seja, cada processador está ligado a 2 processadores vizinhos, formando um quadrado.

Neste quadrado, como mostra a figura II.1, cada vértice corresponde a um processador com memória local associada, denominado elemento de processamento, e as arestas representam as interligações entre esses processadores, que são implementadas através da utilização de memória compartilhada. Desta forma cada processador endereça uma parte como memória local, e outra parte como compartilhada. Esta característica de combinar um Sistema de Memória Distribuída com um Sistema de Memória Compartilhada é que gerou a denominação de Híbrida a esta máquina.

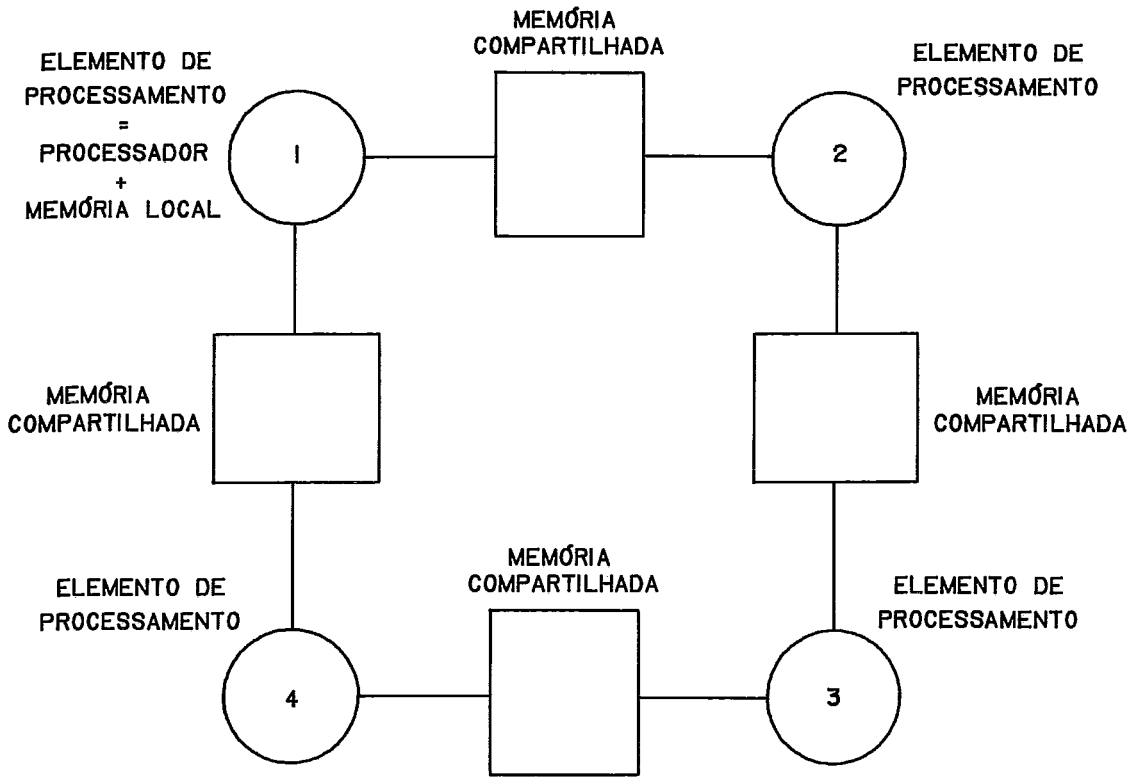


FIGURA II.1 - TOPOLOGIA DA MPH

Em um Sistema de Memória Compartilhada, um dos problemas mais complexos é o conflito de barramento quando dois processadores solicitam o uso da memória ao mesmo tempo. Na MPH esta questão foi solucionada devido às características do microprocessador utilizado na implementação, o 68B09E da empresa MOTOROLA (MOTOROLA INC., 1985). Neste processador o acesso à memória é controlado pelo ciclo do sinal de clock, em um instante bem definido (praticamente durante o tempo em que o sinal de clock está alto). Assim, para que dois processadores possam fazer acesso à mesma memória, sem que ocorra conflito, basta que seus sinais de clock sejam defasados 180 graus, ou seja, quando um processador estiver com o sinal de clock alto (fazendo um acesso à memória), o outro processador está com

o sinal de clock baixo (não fazendo acesso).

II.2 - Organização da Memória

O microprocessador 68B09E pode endereçar até 64K bytes de memória, que foi dividida em 8 bancos de 8K bytes cada, organizados da seguinte forma (figura II.2), SOUZA (1988):

. 3(três) bancos reservados para memória local;

. 4(quatro) bancos utilizados como memória compartilhada;

. 1(um) banco para finalidades diversas.

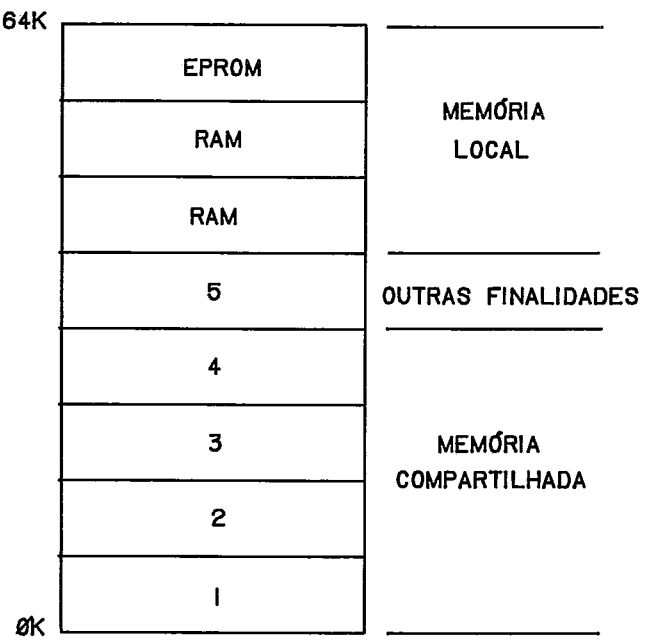


FIGURA II.2 - MAPA DE MEMÓRIA

Dos três bancos de memória local, dois são RAM e um é EPROM. O banco disponível para outras finalidades pode ser usado como área de comunicação com outro processador mais poderoso. Desta forma, a MPH poderia servir como Co-processador Distribuído de Comunicação, ou seja, cada

elemento de processamento possuindo uma ligação com um processador mais poderoso, e as arestas implementando a interligação desses processadores.

A troca de mensagens entre os elementos de processamento da MPH é feita simplesmente escrevendo-se na área de memória compartilhada entre dois elementos vizinhos.

II.3 - Memória Compartilhada

Em uma área de memória compartilhada é possível tanto implementar um esquema de troca de mensagens, quanto permitir a execução, simultânea ou não, de um mesmo trecho de código de programa. A primeira hipótese é atraente no sentido de que a taxa de transferência das mensagens é igual à taxa de acesso à memória local, já que a arquitetura não possui árbitro, sendo portanto bastante alta (da ordem de 16Mbits/s). A segunda hipótese leva em consideração aspectos da utilização eficiente da memória, pois a colocação de um mesmo código em uma área compartilhada diminui o número de cópias deste código em toda a máquina.

Com relação à primeira hipótese, algumas facilidades foram implementadas na MPH. A troca de mensagens entre dois processadores se realiza quando um processador escreve em uma área de memória compartilhada e o outro procesador lê desta mesma área. Para isso, quando um processador escreve na área de memória compartilhada, uma interrupção é gerada para o outro processador. O circuito gerador desta

interrupção conta um tempo entre cada escrita, para que se possa determinar o fim da mensagem, e só então a interrupção é gerada.

Cada banco de memória compartilhada (figura II.3) se constitui de três partições de tamanho selecionável a partir de micro-chaves no circuito de cada processador. As partições superior e inferior podem ter o tamanho 256, 512, 1024, 2048 e 4096 bytes, ficando a partição comum com o restante, quando houver.

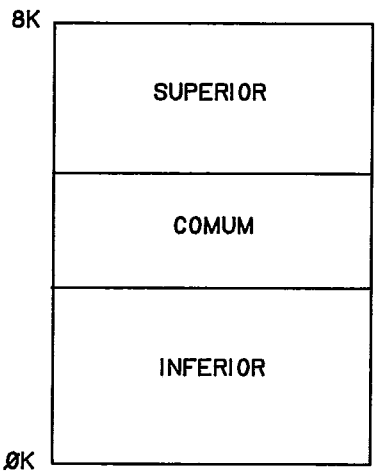


FIGURA II.3 - BANCO DE MEMÓRIA COMPARTILHADA

Escolheu-se, arbitrariamente, que os microprocessadores com sinal de clock positivo utiizem a parte superior dos bancos de memória para enviar a mensagem, e que os microprocessadores com sinal de clock contrário utilizem a parte inferior, como ilustra a figura II.4.

Quando uma interrupção de escrita na área compartilhada é gerada, o processador que a recebe pode reconhecer qual o processador que a gerou através da leitura de um endereço

de memória local, que indica em que banco de memória a mensagem foi escrita. Este endereço de memória local, na realidade, mapeia um grupo de 5 flip-flop's (cada um associado a um banco) e a sua leitura provoca a reiniciação do conteúdo dos flip-flop's para 0.

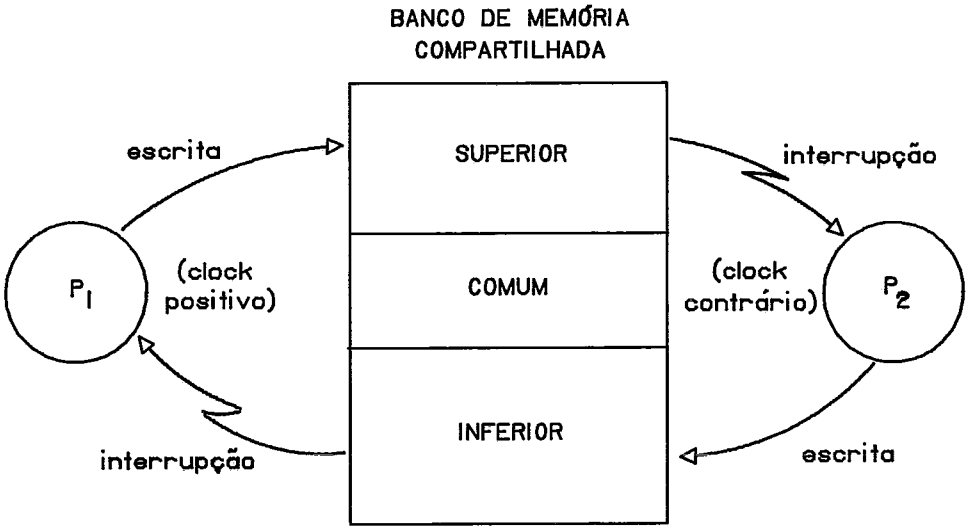


FIGURA II.4 - COMUNICAÇÃO ENTRE PROCESSADORES

II.4 - Relógio de Tempo Real

Na MPH foi utilizado um circuito integrado de relógio para gerar interrupções de contagem de tempo no microprocessador. O circuito integrado escolhido foi o 68B40 da família do microprocessador 68B09E (MOTOROLA INC., 1985). Ele possui três contadores binários de 16 bits, sendo um utilizado para a geração do sinal de interrupção de escrita na área de memória compartilhada.

Os outros dois contadores são utilizados em cascata para a geração do "tick" do relógio de tempo real do

processador. O valor do "tick" pode ser programado no relógio.

II.5 - Estado Atual

Como foi dito na seção II.1, a MPH atualmente está implementada, a nível de protótipo, com 4(quatro) microprocessadores (vértices de um quadrado).

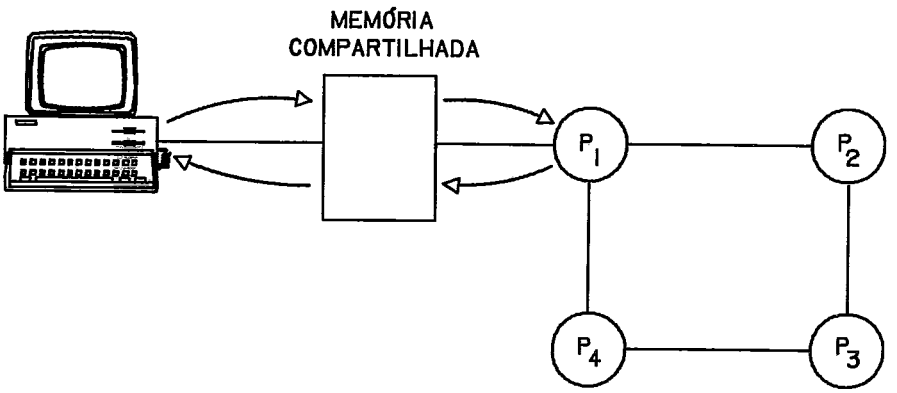


FIGURA II.5 - LIGAÇÃO ENTRE O MICROCOMPUTADOR COLOR-64 E A MPH

A MPH está interligada com um microcomputador COLOR-64, da empresa LZ Informática, que utiliza o mesmo microprocessador da MPH. Esta interligação é feita através da memória compartilhada de um dos seus processadores, tornando o COLOR-64 uma espécie de "front-end-processor" (figura II.5). Os programas e dados a serem carregados na MPH são colocados no COLOR-64 e, através de um utilitário, são transferidos para a MPH. Esta interligação está implementada de forma que, para o COLOR-64, a memória compartilhada é sua.

CAPITULO III

ESPECIFICAÇÃO DO NÚCLEO DO SISTEMA OPERACIONAL DA MPH

III.1 - Introdução

O Sistema Operacional é um conjunto de procedimentos de controle e rotinas de gerenciamento da utilização pelos processos dos recursos existentes na máquina. Um Sistema Operacional deve possuir políticas para a escolha da ordem em que os processos serão atendidos em suas requisições, como também para resolver conflitos de requisições simultâneas.

Um Sistema Operacional pode ser visualizado como um conjunto de camadas, de uma forma hierárquica, DEITEL (1984). Na base da hierarquia está a máquina, como mostra a figura III.1. Na camada imediatamente acima está o Núcleo que, além de controlar a máquina, oferece algumas funções adicionais, que estendem a visão da máquina tanto para o usuário quanto para o próprio Sistema Operacional. Essas funções adicionais são chamadas **Primitivas.** Acima do Núcleo estão os processos do Sistema Operacional que fornecem suporte aos processos dos usuários, como por exemplo o processo de gerenciamento dos dispositivos de entrada e saída. No topo da hierarquia estão os processos dos usuários.

HOLT (1978) define que Núcleo do Sistema Operacional é um módulo fundamental que possui a responsabilidade de escalonar o processador entre os processos, receber

interrupções de E/S e transmití-las ao processo que gerencia o dispositivo de E/S, e suportar a comunicação entre processos. No Núcleo os processos são identificados através dos **Descritores de Processos** que são áreas que contém a sua representação física e lógica.

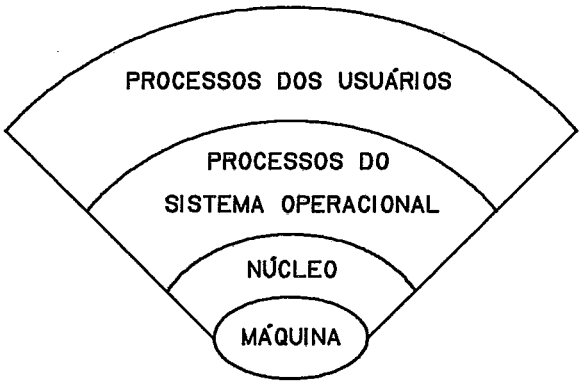


FIGURA III.1 - HIERARQUIA DO SISTEMA OPERACIONAL

O Núcleo também recebe as interrupções do relógio do Sistema. Na verdade, ele é o único módulo que reconhece as interrupções ocorridas na máquina. Fora do Núcleo as interrupções são invisíveis e não afetam a sequência lógica dos processos. Para um processo, tudo se passa como se ele possuisse um processador só para si. A figura III.2 mostra o diagrama de funcionamento de um Núcleo.

A tarefa de atribuir um processador a um processo, para que este possa efetivamente executar, é chamada escalonamento ou despacho de processos e deve ser executada de maneira que o maior número possível de requisições de processos sejam servidos no menor tempo e que todos os processos com mesma prioridade possam usar o processador e

os recursos da máquina por tempo igual.

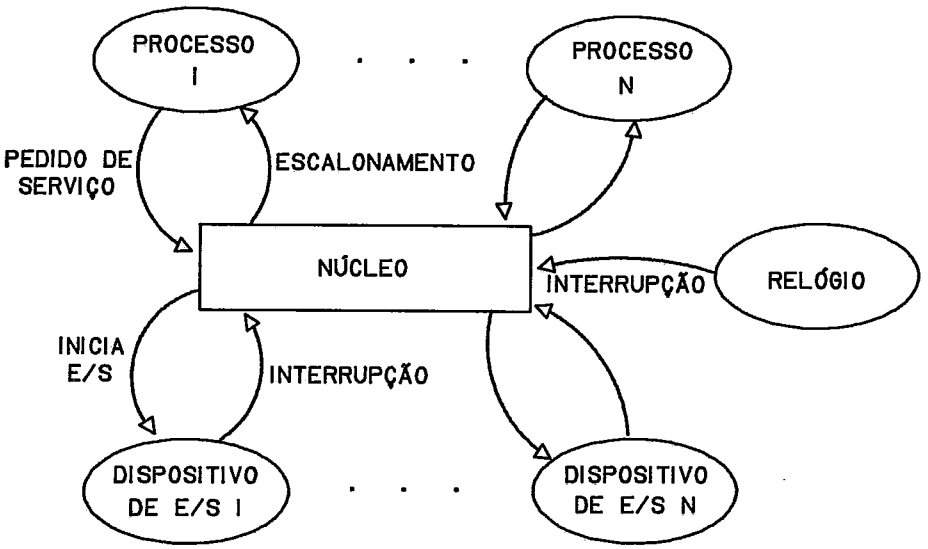


FIGURA III.2 - FUNCIONAMENTO DE UM NÚCLEO

O Núcleo do Sistema Operacional da MPH fornece as ferramentas básicas para que processos possam ser executados. Essas ferramentas são as rotinas de execução de serviços solicitados por processos e o ambiente necessário à execução dessas rotinas. Estabeleceu-se que o Núcleo proveria aos processos os seguintes serviços básicos:

- envio de mensagem de um processo a outro;
- recebimento de uma mensagem por um processo;
- espera de um intervalo de tempo sem execução;
- criação de um processo;
- término do processo que estava executando;
- alocação de UCP aos processos.

O Núcleo é executado sem ser interrompido; portanto as interrupções são inibidas durante o pedido dos serviços dos

processos.

Além desses serviços o Núcleo reconhece a ocorrência de interrupções e as trata de acordo com a especificação da MPH e com a especificação do próprio Núcleo. As interrupções que podem ocorrer são:

. indicação de escrita na área de memória compartilhada;

. "tick" do relógio de tempo real do sistema.

III.2 - Processos Concorrentes

Os processos são os programas que executam em um processador. O processo inclui as instruções armazenadas na memória, registradores, indicadores, área de memória para dados, ou seja, tudo o que é necessário para a execução de uma tarefa ou de um programa. Os processos são concorrentes se eles existem ao mesmo tempo. Processos Concorrentes podem executar completamente independentes ou podem requerer alguma cooperação, DEITEL (1984).

Tipicamente, na MPH os processos são subrotinas ativadas por um comando de execução paralela do tipo "cobegin" ou "parbegin". Esses comandos são definidos sob a forma de pares do tipo "cobegin/coend" ou "parbegin/parend", e delimitam a sequência de instruções que serão executadas em paralelo. No escopo deste trabalho essa sequência só poderá conter chamadas a sub-programas. No entanto, uma sequência pode conter mais de uma chamada a um mesmo sub-programa. A ativação de uma rotina é denominada instância da rotina. Assim, os processos são

instâncias de rotinas. Dentro de um processador pode existir mais de uma instância de uma mesma rotina, porém a área de código para execução não é copiada para cada instância. Cada rotina possui uma área de código que é compartilhada por todas as instâncias.

As rotinas são carregadas na MPH através da conexão entre o microcomputador COLOR-64 e o primeiro processador. Este processador recebe as suas rotinas, e transmite as rotinas dos outros processadores. São criadas estruturas em uma tabela especificando a identificação de cada rotina, sua localização na memória, área de dados que necessita e endereço inicial para a sua execução.

A identificação de um processo inclui três parâmetros: processador no qual executa; rotina que executa; e, instância que o representa. Dessa forma, a combinação desses três parâmetros produz uma única identificação dos processos de toda a máquina.

Quando um processo é criado, ele inicia e termina sua execução no mesmo processador. Esta definição é dada por que a divisão de processos entre processadores é feita durante a compilação do programa. Não é realizado na máquina o balanceamento de carga dos processadores.

Em cada processador, executam um ou mais processos que podem cooperar entre si trocando mensagens e que podem concorrer pelo uso do relógio de tempo real, por tempo de processador, e pelo espaço de memória. Processos de um

processador também podem cooperar com outros processos executando em outros processadores através da troca de mensagens através da área de memória compartilhada.

Os processos são identificados através dos **Descritores de Processos** que contêm todas as informações necessárias ao Núcleo para sua manipulação. Essas informações, são apresentadas abaixo:

- a identificação do processo;
- o processador no qual este processo executa;
- identificador do processo que o criou;
- número de processos que ele criou;
- localização do código do processo e a quantidade de memória que ocupa;
- localização da área de dados e da área de pilha do processo e seus tamanhos;
- o estado atual do processo;
- identificação do processo com o qual esteja se comunicando;
- endereço da mensagem a transferir;
- uma área para guardar o contexto do processo;
- um apontador para a lista de mensagens disponíveis para o processo;
- tempo para espera sem execução;
- fatia de tempo para o processo.

A identificação do processo, o processador no qual ele executa e a localização e tamanho das suas áreas de código, dados e pilha, são informações que identificam e localizam os processos. As outras informações serão apresentadas ao

longo deste capítulo.

III.3 - Estados dos Processos

Durante a execução os processos podem assumir os seguintes estados:

- **executando;**
- **bloqueado;**
- **pronto** para executar.

Um processo está no estado **executando** quando efetivamente utiliza o processador. No estado **bloqueado** o processo está esperando a ocorrência de algum evento. Um processo está **pronto** para executar quando está esperando que o processador lhe seja atribuído, PETERSON (1985).

Na MPH um processo fica **bloqueado** quando:

- pede a espera por um intervalo de tempo sem execução;
- pede o recebimento de uma mensagem que ainda não chegou;
- cria novos processos, aguardando que eles terminem;
- pede o envio de uma mensagem, porém não existe espaço suficiente para armazenar esta mensagem.

Assim que qualquer uma dessas condições deixar de existir, o processo é desbloqueado e passa para o estado **pronto** para executar.

Um processo só é executado quando o processo que estava executando anteriormente for **bloqueado** ou for retirado da

execução, e ele for o novo processo escolhido para executar. A escolha de um processo para executar é realizada pelo Escalonador de Processos. O diagrama de Estados dos Processos na MPH está mostrado na figura III.3.

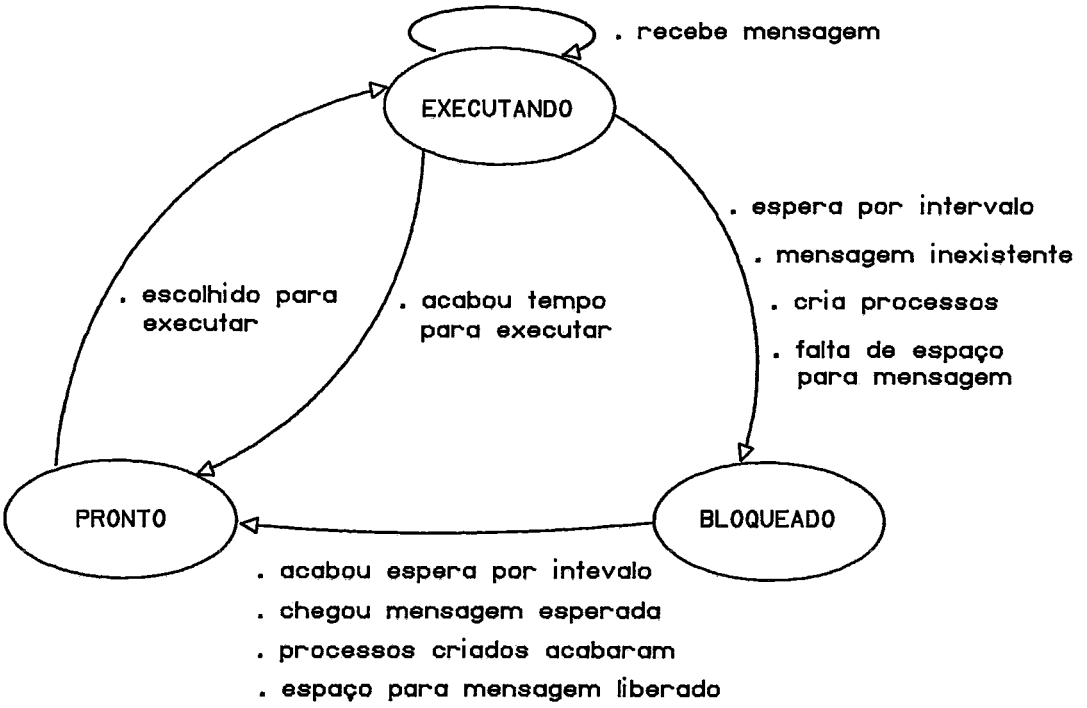


FIGURA III.3 - ESTADOS DOS PROCESSOS

O Núcleo do Sistema Operacional da MPH possui listas onde os processos são colocados de acordo com seus estados:

- **Processos Prontos** - lista que contém os **Descritores de Processos** que estão à espera de processador para executar;
- **Processos Bloqueados por Temporização** - lista ordenada por intervalo de espera sem execução;
- **Processos Bloqueados à Espera de Mensagem** - lista contendo todos os processos à espera de mensagem;
- **Processos Bloqueados pela Criação de Novos Processos** - lista contendo os processos à espera do término dos processos criados;

- **Processos Bloqueados à Espera de "Buffers"** - lista com processos à espera de espaço para envio de mensagens dentro de um processador;
- **Processos Bloqueados à Espera de Área Compartilhada** - lista de processos esperando liberação de área compartilhada para comunicação externa.

As duas últimas listas serão apresentadas e comentadas na seção III.8.3.

III.4 - Escalonamento de Processos na MPH

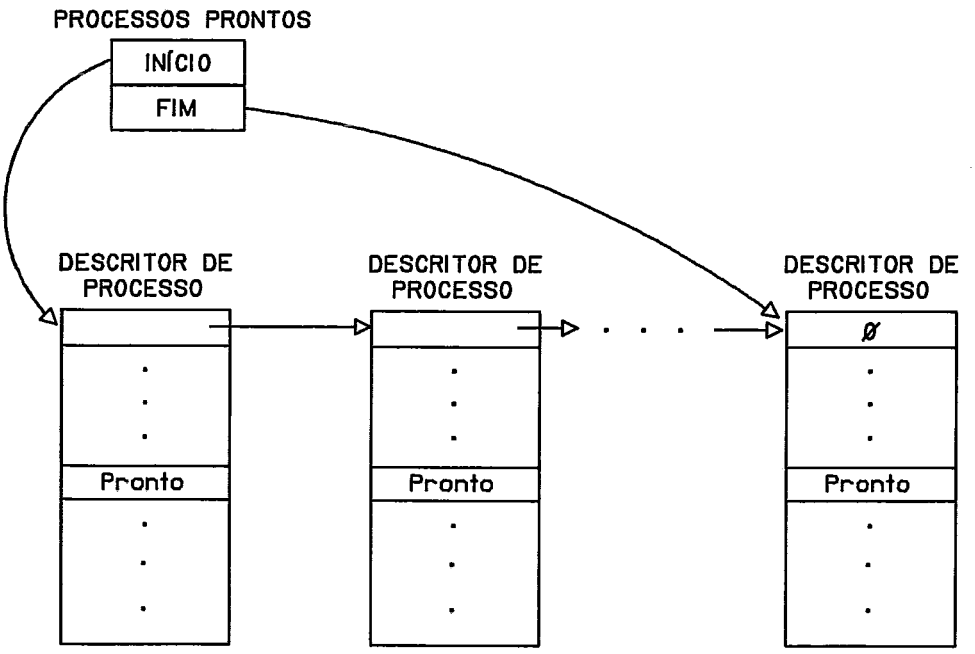


FIGURA III.4 - LISTA DE PROCESSOS PRONTOS

Para a MPH foi especificado que os processos não possuem prioridade. Cada um recebe um tempo para executar ("time-slice"). Se este tempo não for suficiente para o processo terminar, então ele vai para o fim da lista de **Processos Prontos** e espera uma nova chance para executar.

Durante a execução de um processo as interrupções estão habilitadas e são tratadas normalmemte. A lista de **Processos Prontos** (figura III.4), para simplificar a implemetação, é do tipo FIFO e o processo que está executando é o primeiro da lista. Quando ele é interrompido, o processo seguinte na lista passa a ser o primeiro e este é escalonado para executar, recebendo também um tempo para executar. Este tempo será especificado estaticamente no início da implementação e, de acordo com o comportamento do sistema, este tempo será ajustado para que todos os processos tenham tratamento uniforme.

III.5 - Criação e Término de Processos

O Núcleo permite que um processo crie novos processos dinamicamente. Quando isto ocorre, o processo que estava executando e pediu este serviço é **bloqueado** e seus processos filhos passam a executar. O processo é colocado na lista de **Processos Bloqueados pela Criação de Novos Processos** (figura III.5), e o número de processos que ele criou é colocado em seu descritor.

O Núcleo possui uma lista de **Descritores de Processos Vazios** que podem ser alocados quando processos são criados. O descritor de um processo é preenchido com as informações necessárias e a identificação de seu processo pai. Quando um processo acaba de executar, ele gera uma requisição ao Núcleo para seu término. Neste momento, o descritor deste processo é verificado em relação às suas pendências. Podem existir mensagens que não foram consumidas, por exemplo, e

essas mensagens devem ser ignoradas para a correta terminação. O **Descritor de Processo** correspondente é liberado e volta para a lista de **Descritores Vazios.** O processo pai do processo que terminou deve ser notificado da ocorrência.

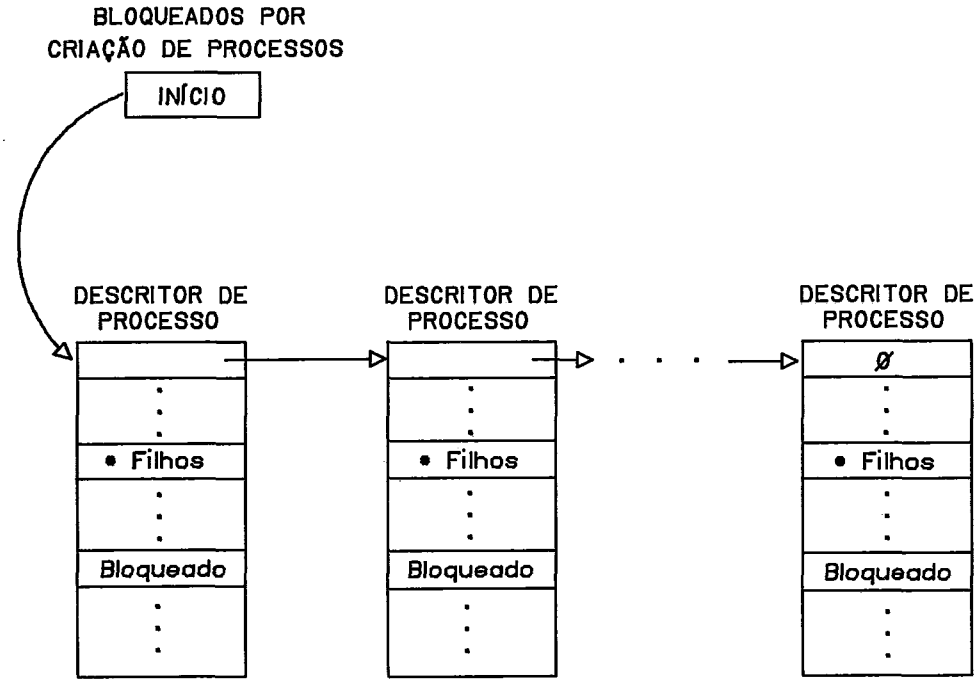


FIGURA III.5 - LISTA DE PROCESSOS BLOQUEADOS PELA CRIAÇÃO DE NOVOS PROCESSOS

Quando todos os processos filhos acabarem o processo pai é desbloqueado, sendo inserido no fim da lista de **Processos Prontos.** Os primeiros processos que são criados na máquina possuem como identificador do processo pai o valor 0, ou seja, não possuem pai. Um processo filho também pode criar novos processos, formando uma outra geração e assim por diante.

Em cada elemento de processamento o Sistema Operacional mantém uma tabela com a localização dos processos. Esta **Tabela de Processos** contém os identificadores de todos os

processos ativos na máquina e em qual processador esses processos residem. Quando um novo processo é criado o Núcleo atualiza a **Tabela de Processos** para incluí-lo e quando o processo termina ele é excluído da tabela. A organização desta tabela está mostrada na figura III.6.

PROCESSADOR	IDENTIFICADORES DOS PROCESSOS			
1	257	258	Ø	Ø
2	Ø	Ø	Ø	Ø
3	Ø	Ø	Ø	Ø
4	Ø	Ø	Ø	32767

FIGURA III.6 - FORMATO DA TABELA DE PROCESSOS

Um processo pode criar novos processos em qualquer elemento de processamento, isto é, no elemento em que reside e/ou em outro elemento da máquina.

III.6 - Mensagens de Sistema

Uma classe de mensagens especiais que tem como destino o Núcleo do Sistema Operacional foi especificada. Estas mensagens possuem como endereço de destino o valor 0.

As mensagens de destino 0 podem ser de quatro tipos diferentes, cujos tamanhos variam, de acordo com a função que possuem :

- carregamento de uma rotina em um elemento de processamento;
- criação de um processo em um elemento de processamento;
- inclusão de um processo nas **Tabelas de Processos** de todos os elementos de processamento;
- aviso de término de um processo e sua consequente exclusão das **Tabelas de Processos** de todos os elementos de processamento.

Os dois primeiros tipos de mensagem são direcionadas para um elemento de processamento específico, isto é, uma rotina é carregada em um determinado elemento e um processo será criado em um elemento específico. Os processos estão sempre associados a um único elemento, portanto, se são criados em um elemento deverão executar até o término no mesmo elemento. Os dois últimos tipos de mensagem listados acima, devem ser destinadas a todos os elementos, criando-se assim uma mensagem de difusão ("broadcast").

De acordo com os mecanismos utilizados para roteamento de mensagens entre elementos de processamento e mensagens de difusão em uma máquina com topologia hipercúbica, foram fixadas as rotas para troca de mensagens entre elementos de processamento não vizinhos, HAYES (1989). Quando um elemento deseja enviar mensagem a um elemento não vizinho, a mensagem é enviada primeiramente ao seu vizinho que faz parte do cubo de dimensão inferior e deste para o elemento destino. Por exemplo, o elemento 1 envia mensagem para o elemento 4 para atingir o elemento 3 e o elemento 3 envia mensagem para o elemento 2 para atingir o elemento 1. A

mensagem de difusão é implementada de forma que o elemento que gera a mensagem, envia 2 mensagens e somente um dos elementos é que envia mais uma mensagem. Com este método é possível se garantir que todos os elementos receberam e reconheceram a mensagem.

A mensagem de carregamento de rotina inicialmente é gerada no microcomputador COLOR-64 e é direcionada ao primeiro processador. Este processador pode gerar as mensagens de carregamento de rotina para os outros processadores ou simplesmente rotear as mensagens originadas no COLOR-64 que são endereçadas aos outros processadores da máquina. As informações necessárias ao carregamento de uma rotina são:

. o nome da rotina;
. o endereço e o tamanho da memória onde o código será carregado;
. o endereço e o tamanho da área de pilha que a rotina precisa;
. o endereço inicial de exeução.

Essas informações são armazenadas em uma estrutura no Núcleo, chamada **Tabela de Rotinas**, onde todas as rotinas carregadas na memória são identificadas. Quando um processo for criado a rotina correspondente já estará carregada na memória do elemento de processamento.

Uma mensagem de criação de um processo em um elemento de processamento será enviada sempre que um processo desejar um ou mais de seus processos filhos em outro

elemento. As informações contidas nesta mensagem são:

. o nome da rotina;

. o número da instância da rotina;

. identificador do processo pai.

Esta última informação é necessária primeiramente para preencher o **Descritor do Processo** filho e mais tarde no momento do seu término quando o processo pai tiver que ser avisado.

A mensagem de inclusão de um processo nas **Tabelas de Processos** dos elementos de processamento é enviada logo após a criação de um processo. A mensagem deve ser difundida por toda a máquina para que todos os elementos possam atualizar suas tabelas. As informações que circulam neste tipo de mensagem são:

. elemento que gerou a mensagem, e consequentemente, onde o processo reside;

. o identificador do novo processo.

Quando um processo termina, todos os elementos devem receber uma mensagem para excluí-lo de suas tabelas e informar ao processo pai o seu término, se este não reside no mesmo elemento que o processo pai. A mensagem de término deve ser difundida por toda a máquina e o seu conteúdo é:

. elemento onde o proceso terminou, ou seja, o gerador da mensagem;

. o identificador do processo que terminou;

. o identificador do processo pai.

III.7 - Relógio do Sistema

Como foi dito na seção II.4, a MPH possui 2 contadores cascateados para gerar a interrupção de "tick" do relógio de tempo real para o Núcleo do Sistema Operacional. Definiu-se que esta interrupção será utilizada por duas funções do Núcleo. A primeira função é a de delimitar a fatia de tempo que será alocada a um processo ("time-slice"). Um valor de "time-slice" é colocado no campo de fatia de tempo do **Descritor do Processo** que é decrementado a cada ocorrência da interrupção de "tick". Quando o valor deste campo chega a zero (0), o processador é retirado do processo, e este é colocado no fim da lista de **Processos Prontos**. Quando o processo voltar à execução, um novo valor é carregado neste campo. Se, por qualquer motivo, o processo for **bloqueado** e o valor deste campo ainda não chegou a zero, o valor fica guardado em seu descritor. No momento em que o processo voltar à execução, o valor antigo do campo é considerado e passa a ser decrementado, ou seja, se o valor do campo não for igual a zero, nenhum novo valor é carregado no campo. Com a aplicação desta técnica, é possível garantir um tempo de resposta razoável e prevenir que o sistema fique dedicado a um processo que possa estar em um "loop" infinito.

A segunda função implementada através do relógio é a espera por intervalo de tempo pelos processos ("delay"). Quando um processo solicita esse tipo de serviço ao núcleo, o tempo desejado é manipulado e carregado em um campo de seu descritor. O processo, então, é **bloqueado** e inserido na

lista de **Processos Bloqueados por Temporização** (figura III.7). Esta lista é ordenada pelos intervalos de espera dos processos de forma que, quando um processo pede uma temporização ao núcleo e especifica o intervalo de tempo, a lista é percorrida desde o início e os intervalos de espera contidos nos **Descritores de Processos** da lista vão sendo somados e comparados com o intervalo desejado pelo processo. O **Descritor de Processo** deve ser inserido na lista entre dois descritores, quando o intervalo desejado pelo processo for maior ou igual ao valor acumulado da lista até o descritor anterior, e menor que o valor acumulado da lista até o descritor posterior. O valor que passa a existir no campo de intervalo de espera do **Descritor do Processo** é igual à diferença entre o intervalo desejado pelo processo e o valor acumulado da lista até o descritor anterior. Portanto, os valores contidos nos campos de intervalo de espera dos processos desta lista estão relacionados entre si. Quando a interrupção de "tick" do relógio de tempo real ocorre, o valor do campo do primeiro descritor da lista é decrementado e se este valor chegar a zero, o descritor do processo é retirado desta lista e é colocado no fim da lista de **Processos Prontos**. O segundo descritor da lista torna-se o primeiro e o valor de seu campo é que será atualizado pela interrupção, e assim por diante.

A geração desta interrupção é desabilitada sempre que o Núcleo passa a executar, tanto pelo pedido de um serviço de um processo quanto pela ocorrência de alguma interrupção. Desta forma, o tempo que um processo tem para executar é

congelado não sendo gasto com as tarefas do Núcleo.

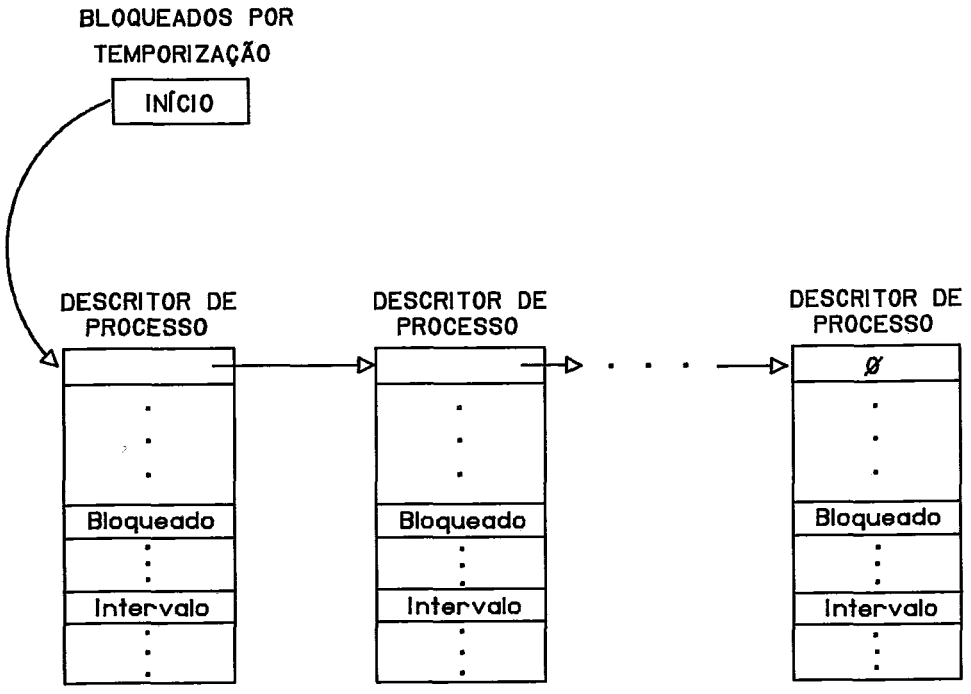


FIGURA III.7 - LISTA DE PROCESSOS BLOQUEADOS POR TEMPORIZAÇÃO

III.8 - Comunicação entre Processos na MPH

Na MPH os processos interagem através da troca de mensagens. Definiu-se que as **Primitivas** do Núcleo que executam esta função são **Envia** e **Recebe**. A primitiva **Envia** é solicitada quando um processo deseja enviar uma mensagem a outro processo. A primitiva **Recebe** é ativada quando um processo deseja receber mensagem de outro processo. Quanto à sincronização, a primitiva **Envia** é **não-bloqueante**, ou seja, a solicitação desta primitiva não bloqueia a execução do processo, e a primitiva **Recebe** é **bloqueante**; portanto, o processo fica **bloqueado** se não existir mensagem para ele. O endereçamento das mensagens dentro do Núcleo é explícito e simétrico; assim, os nomes dos processos origem e destino

da mensagem estão referenciados explicitamente. As mensagens têm tamanho variável. O formato das mensagens que transitam na máquina está mostrado na figura III.8. Porém, os processos somente passam como parâmetro nas **Primitivas** uma estrutura em que o primeiro campo é o tamanho da mensagem e o resto é a mensagem propriamente dita.

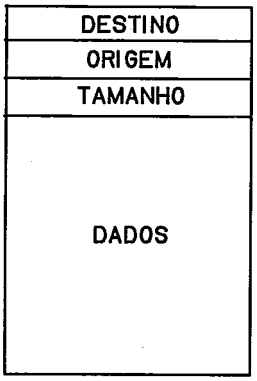


FIGURA III.8 - FORMATO DAS MENSAGENS

Já é possível se determinar, com isso, quais os parâmetros necessários às **Primitivas Envia** e **Recebe**. A primitiva **Envia** tem como parâmetros:

- processo destino da mensagem;
- endereço da estrutura que contem o tamanho e a mensagem.

A primitiva **Recebe** tem como parâmetros:

- endereço da estrutura onde o Núcleo deve colocar o tamanho e a mensagem;
- processo origem da mensagem.

Na **Primitiva Envia** o usuário não precisa passar como parâmetro a origem da mensagem, pois o Núcleo irá fornecer o identificador do processo origem, que é o próprio processo que está executando no momento. Na **Primitiva Recebe** o processo receptor pode escolher de qual processo emissor quer receber uma mensagem.

A identificação dos processos sempre é feita considerando-se três parâmetros: o processador; a rotina; e a instância da rotina. Esta forma de identificação de um processo deve também ser mantida na ativação das **Primitivas Envia** e **Recebe**.

Pode-se considerar que na MPH existem duas vias de comunicação. Se os procesos que se comunicam executam no mesmo processador, utiliza-se uma via interna e se, por outro lado, os processos estão em processadores diferentes, uma via externa é utilizada. Internamente foi definido que um "pool" de "buffers" estivesse disponível para a comunicação. Externamente a troca de mensagens é realizada através de bancos de Memória Compartilhada, como foi visto na seção II.3.

Como já foi mencionado, a configuração corrente da MPH inclui 4 elementos de processamento formando um quadrado (figura II.1). Por essa característica, um elemento está interligado diretamente com os elementos vizinhos. Se um processo de um elemento de processamento desejar se comunicar com um processo em um elemento não vizinho, a

mensagem deverá percorrer uma rota através dos elementos vizinhos (figura III.9).

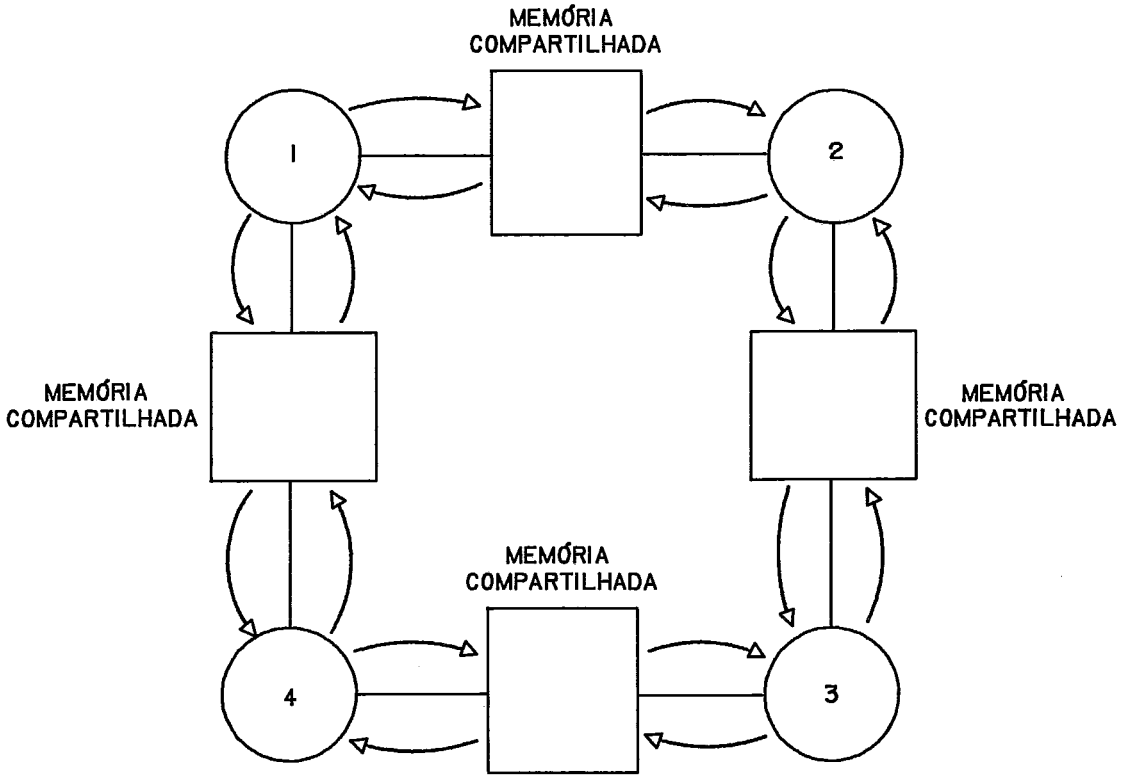


FIGURA III.9 - ROTA DAS MENSAGENS

Para que o Núcleo tivesse conhecimento da localização dos processos em relação aos elementos de processamento, foi incluída em cada elemento uma **Tabela de Processos.** O caminho que uma mensagem deve seguir para alcançar o elemento que contém o processo destino é determinado por uma **Tabela de Rotas** residente em cada elemento. Esta tabela contém o número do elemento intermediário que promoverá a rota para o elemento de processamento destino.

III.8.1 - Comunicação Dentro de um Elemento de Processamento

Em cada elemento de processamento foi definido um "pool" de "buffers" onde as mensagens podem ser colocadas pela **Primitiva Envia não-bloqueante** e retiradas pela **Primitiva Recebe.** Sabendo-se que um "buffer" tem tamanho fixo e as mensagens têm tamanho variável, o "buffer" foi definido com três campos (figura III.10) :

- um apontador para um outro "buffer" de forma a encadeá-los em uma lista, o campo **Ponteiro;**
- um encadeamento para outro "buffer" que contém o restante da mensagem, quando a mensagem não couber em um único "buffer", o campo **Cadeia;**
- área de mensagem com um tamanho fixo, o campo **Mensagem.**

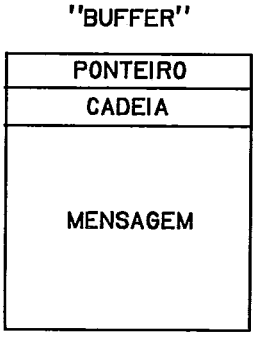


FIGURA III.10 - "BUFFER" DE MENSAGEM

No início todos os "buffers" estão encadeados em uma lista de **"Buffers" Livres.** No momento em que um processo ativa a **Primitiva Envia,** o Núcleo consulta a **Tabela de Processos** e, por exemplo, descobre que o processo destino é local e, portanto, a mensagem é interna. Em seguida, o

Núcleo aloca um "buffer" da lista de **"Buffers" Livres** e guarda a mensagem. Se for necessário, outros "buffers" são alocados e encadeados pelo campo **Cadeia.**

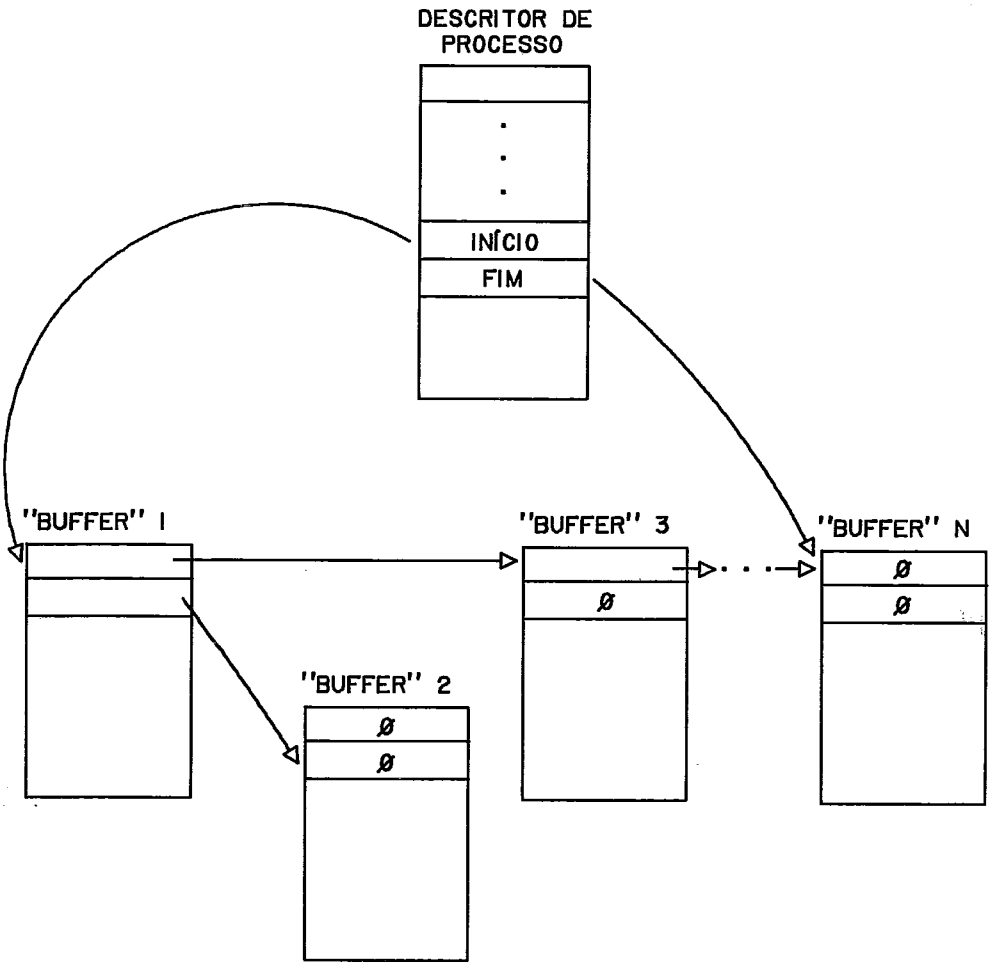


FIGURA III.11 - LISTA DE MENSAGENS DISPONÍVEIS

No **Descritor de Processo** existe um campo de apontadores para uma lista de mensagens recebidas. Esta lista é chamada **Mensagens Disponíveis.** Os "buffers" de mensagem são encadeados nesta lista, com o apontador de início indicando o primeiro "buffer" de mensagem e o apontador fim indicando o último, como mostra a figura III.11. Deve-se notar que o campo **Ponteiro** do "buffer" é utilizado para interligar "buffers" da lista, enquanto que o campo **Cadeia** interliga

"buffers" de uma mesma mensagem.

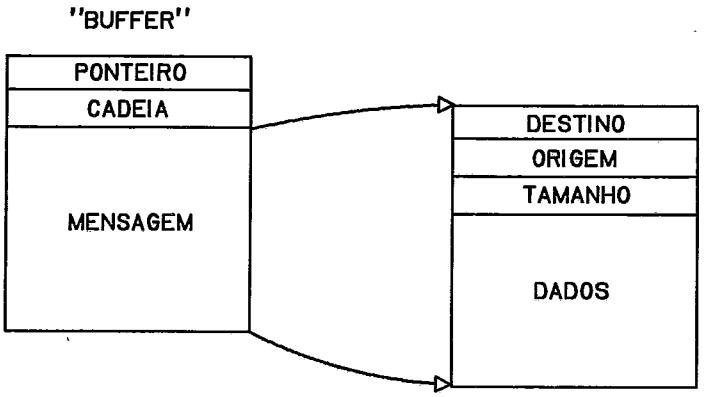


FIGURA III.12 - CONTEÚDO DE UM "BUFFER" DA LISTA DE MENSAGENS DISPONÍVEIS

A função da **Primitiva Recebe** é consultar a lista de **Mensagens Disponíveis** procurando por uma mensagem que foi originada por um processo específico. Isto é possível pois cada "buffer" encadeado na lista contém no campo **Mensagem** o formato da mensagem (figura III.12). Se a lista de **Mensagens Disponíveis** estiver vazia (Início = Fim = 0) ou se não existe nenhuma mensagem na lista, originada no processo especificado, o processo que requisitar a primitiva é **bloqueado** e inserido na lista **Processos Bloqueados à Espera de Mensagem**. Nesta lista (figura III.13) os processos são inseridos no início e retirados aleatoriamente, ou seja, assim que a mensagem desejada for recebida. Dois campos no **Descritor de Processos** guardam qual deve ser o processo remetente da mensagem e o endereço onde o tamanho e a mensagem devem ser devolvidos. Quando finalmente a mensagem for recebida, o processo é desbloqueado, retirado da lista de **Processos Bloqueados à Espera de Mensagem** e inserido no fim da lista de **Processos**

Prontos. A mensagem, então, é escrita no endereço determinado no **Descritor de Processo** e os "buffers" são liberados, voltando para a lista de "buffers" livres.

Como foi mostrado, toda a manipulação de "buffers" e formatação das mensagens, fica a cargo do Núcleo, não necessitando que os processos tenham conhecimento disto.

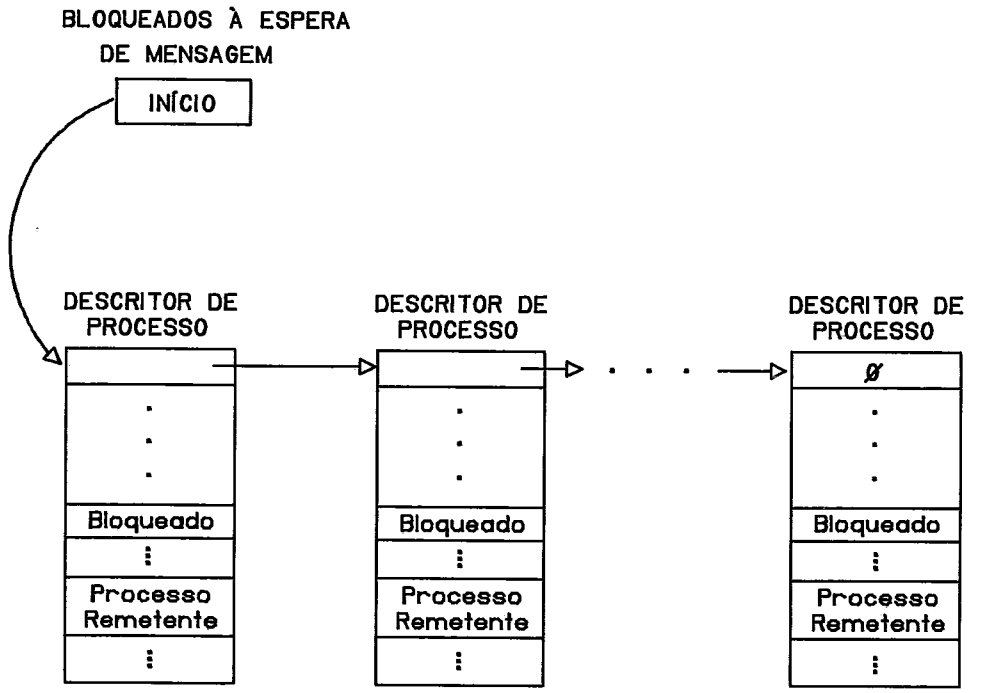


FIGURA III.13 - LISTA DE PROCESSOS BLOQUEADOS À ESPERA DE MENSAGEM

III.8.2 - Comunicação Entre Elementos de Processamento

Quando é preciso que a comunicação se efetue entre elementos de processamento, o banco de Memória Compartilhada é utilizado. Algumas características da arquitetura e das soluções do "hardware" da MPH simplificaram muito a manipulação dos bancos de Memória

Compartilhada. Em primeiro lugar os bancos foram divididos em três partes (superior, comum e inferior), ficando cada partição com uma direção específica (figura II.4), ou seja, cada elemento somente envia mensagem para outro elemento através de uma partição e somente recebe mensagem através de outra partição. Com isso, não há necessidade de se aplicar o conceito de **exclusão mútua**, pois as áreas de escrita dos elementos são distintas. Em segundo lugar, uma interrupção é gerada sempre que um elemento escrever na área que lhe é permitida. Desta forma nenhum processo precisa ficar verificando se chegou alguma mensagem, evitando a espera ocupada.

Definiu-se que as partições do banco nas quais ocorre troca de mensagem, a superior e a inferior,sejam tratadas como "buffers" circulares. Um "buffer" circular possui dois apontadores para o controle de sua ocupação, como mostra a figura III.14.

O apontador C indica o próximo item a ser consumido, se referenciando, assim, aos ítens cheios. O apontador P indica o próximo ítem a ser utilizado pelo produtor, mostrando os ítens vazios. Esses apontadores devem sempre se mover em um único sentido, que de acordo com a figura III.14 é o sentido horário. O apontador P nunca pode ultrapassar o apontador C, porque senão o produtor utilizaria itens já ocupados.

Os valores dos apontadores P e C variam de 0 a (maxbuf - 1), ou seja, número de itens no "buffer" menos 1. Assim,

temos:

. P = C = 0, quando o "buffer" está vazio;

. P - C = maxbuf - 1, quando o "buffer" está cheio.

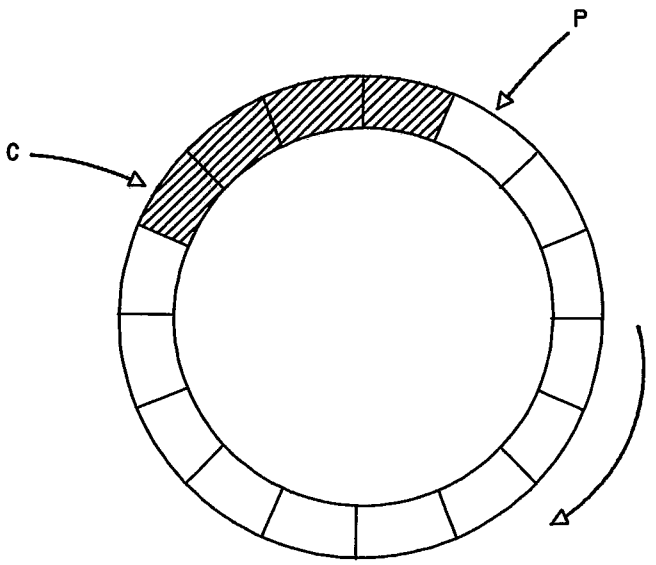


FIGURA III.14 - "BUFFER" CIRCULAR

A adição e remoção de itens do "buffer" se processa da seguinte forma:

Produtor	Consumidor
se "buffer" não está cheio	se "buffer" não está vazio
adiciona item no "buffer"	remove item do "buffer"
P = P + 1	C = C + 1

Todas as operações realizadas nos apontadores P e C devem levar em consideração o número de itens do "buffer" (utilizando a operação módulo de maxbuf).

Esses apontadores devem estar disponíveis para os Núcleos dos dois elementos de processamento envolvidos na comunicação, portanto também foram colocados na área

compartilhada. O formato da área está mostrado na figura III.15 e os apontadores receberam o nome de **Última Posição Cheia** e **Última Posição Vazia.**

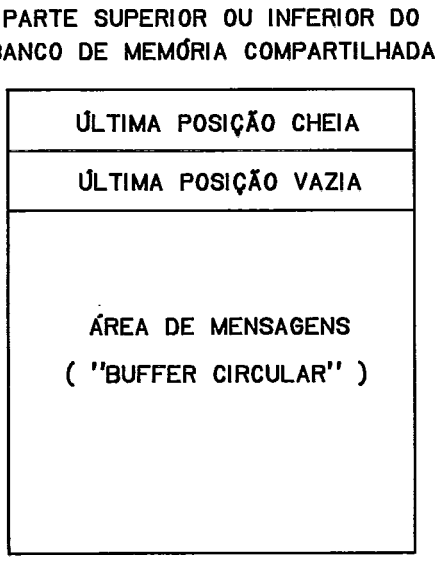


FIGURA III.5 - ÁREA DE MEMÓRIA COMPARTILHADA

Quando o Núcleo recebe uma requisição de envio de mensagem, a **Tabela de Processos** é consultada e se o processo existir em outro elemento de processamento que não o local, então uma nova tabela é verificada. A **Tabela de Rotas** contém a rota para se alcançar o elemento onde o processo destino existe. Com este valor, o Núcleo determina qual o banco e partição que devem ser acessados. A mensagem é escrita na área de Memória Compartilhada (de acordo com o formato na figura III.8), os apontadores da área são atualizados e uma interrupção é gerada para o elemento que receberá a mensagem. O processo que requisitou o envio da mensagem volta a executar normalmente.

A interrupção de recepção de mensagem em um elemento ativa a execução de seu Núcleo, interrompendo momentaneamente o processo que estava executando. A primeira ação do Núcleo é verificar se o processo destino da mensagem existe neste elemento ou apenas deve-se fazer o roteamento da mensagem. A consulta à **Tabela de Processos** indica se o processo existe neste elemento ou em outro. Como a MPH está construída no formato de um quadrado, a mensagem sofrerá, no máximo, um roteamento (vide figura III.9). Se a mensagem precisa ser roteada, ela é simplesmente copiada de uma área compartilhada para outra, conservando o formato montado pelo elemento de processamento que a enviou. Os apontadores das duas áreas são atualizados. Se a mensagem possuir como destino um processo do próprio elemento, então ela deve ser guardada em algum lugar, pois a área compartilhada deve ser liberada o mais rápido possível.

Se ainda existirem mais mensagens na área, elas devem ser tratadas, ou seja, a área compartilhada deve ser lida até que esteja vazia ou até que ocorra uma exceção.

A área de "buffers" interna de um elemento será utilizada também para esta finalidade. Assim, os "buffers" são alocados, a mensagem é copiada e a área compartilhada é liberada. Se o processo destino estiver **bloqueado**, justamente à espera de uma mensagem deste processo origem, então ele pode ser desbloqueado. Neste caso, a mensagem não é colocada em "buffer"; ela é copiada diretamente para a área indicada no **descritor de processo**. O processo, então, é retirado da lista de **Processos Bloqueados à Espera de**

Mensagem e é inserido no fim da lista de **Processos Prontos**. Se, por outro lado, o processo estiver **bloqueado** à espera de uma mensagem de outro processo origem ou, ainda, se não estiver **bloqueado**, a mensagem é copiada nos "buffers" e encadeada na lista de **Mensagens Disponíveis** no **descritor de processo**.

III.8.3 - Tratamento de Exceções

No Núcleo da MPH pode-se chegar a duas condições de exceção:

- os "buffers" internos acabaram;
- a área de Memória Compartilhada está cheia.

Essas condições levam a um tratamento especial dos processos que as provocaram, para que todo o sistema não seja degradado.

A condição de exceção provocada quando os "buffers" internos acabam, ocorre em duas situações:

- um processo requisitou um envio de mensagem para um processo existente no mesmo elemento de processamento e o Núcleo não consegue alocar os "buffers" suficientes para a mensagem;
- a rotina de tratamento de recepção de mensagem não consegue alocar "buffers" para guardá-la na lista de **Mensagens Disponíveis** do processo destino.

Na primeira situação, o processo requisitante deve ser **bloqueado** e inserido na lista de **Processos Bloqueados à Espera de "Buffer"** (figura III.16). Os **Descritores de**

Processo são sempre inseridos no fim da lista. O **Descritor de Processo** é preenchido com a informação do processo destino e do endereço da mensagem a ser enviada. Quando alguma área é liberada a lista é percorrida desde o início e o primeiro processo, na ordem, cuja mensagem couber na área liberada, será desbloqueado, retirado desta lista e colocado na lista de **Processos Prontos.**

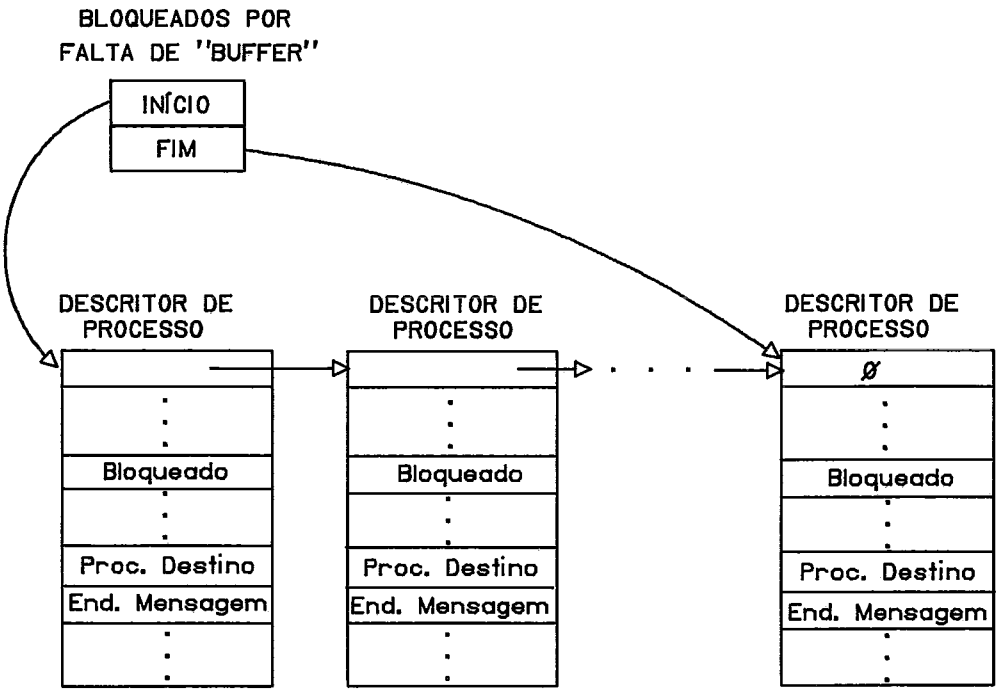


FIGURA III.16 - LISTA DE PROCESSOS BLOQUEADOS À ESPERA DE "BUFFER"

Na segunda situação, a única solução é não liberar a área compartilhada até que existam "buffers" disponíveis para armazenar a mensagem. Quando um ou mais "buffers" forem liberados as mensagens que estiverem na área

compartilhada à espera de "buffers" poderão ser guardadas e a área compartilhada liberada.

Quando os buffers são liberados a ordem de atendimento é: primeiro, se possível, a área compartilhada e depois, se possível, atender à lista de **Processos Bloqueados à Espera de "Buffer"**. Isto se deve à necessidade de liberar a área compartilhada, sob pena de degradação do sistema.

Para que a área de Memória Compartilhada seja liberada rapidamente, é necessário que as mensagens sejam retiradas diretamente da área (quando os processos destino já estão **bloqueados** à espera de mensagem) ou sejam guardadas nos "buffers" para serem consumidas posteriormente.

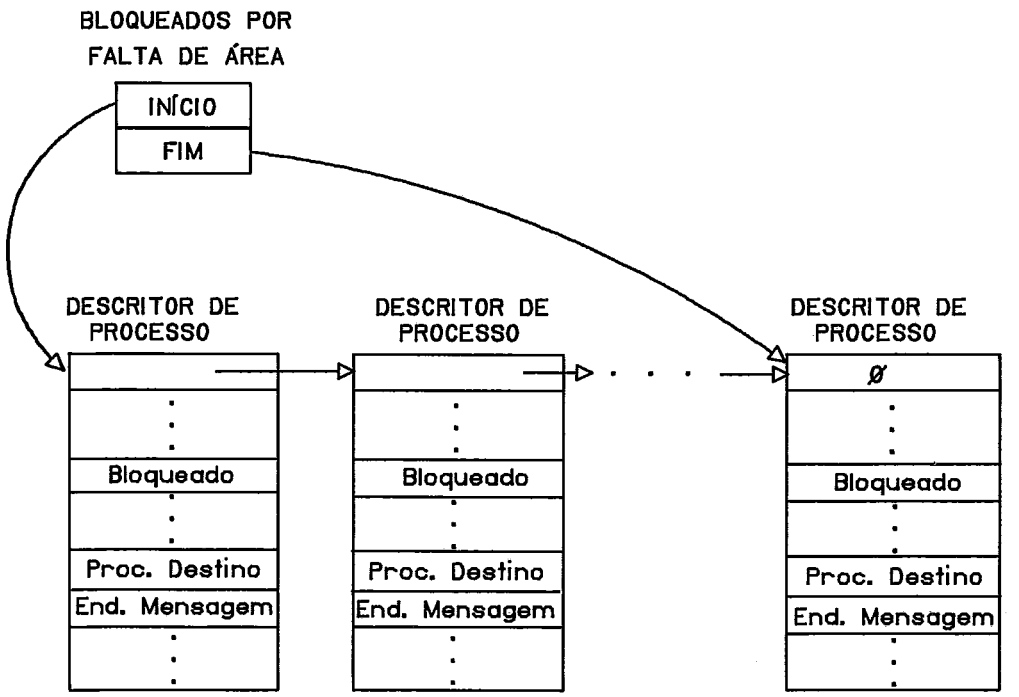


FIGURA III.7 - LISTA DE PROCESSOS BLOQUEADOS À ESPERA DE ÁREA COMPARTILHADA

Mais uma vez, se os "buffers" internos acabarem, uma condição de exceção é criada e um tratamento especial deve ser executado. Nesta situação, o processo que pediu um envio de mensagem e verificou que a área está cheia é **bloqueado** e colocado na lista de **Processos Bloqueados à Espera de Área Compartilhada** (figura III.17). Nesta lista os **Descritores de Processo** são inseridos no fim e retirados, sempre que possível, do início. O **Descritor do Processo** passa a armazenar o identificador do processo destino e o endereço da mensagem a ser enviada. Se o processo da lista não puder ser atendido porque os "buffers" liberados são insuficientes para guardar a mensagem, os próximos processos são analisados na ordem em que se encontram na lista.

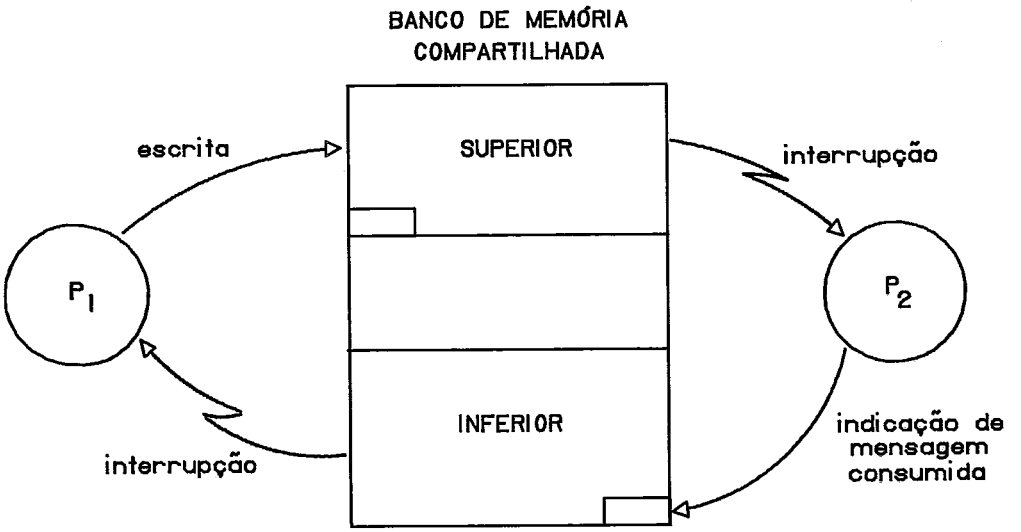


FIGURA III.18 - COMUNICAÇÃO COM INTERRUPÇÃO-PARA-TRÁS

É necessário que exista um mecanismo para indicação de que alguma mensagem foi consumida, significando que alguma parte da área compartilhada foi liberada. O ideal seria que existisse uma **interrupção para trás**, isto é, um processo coloca uma mensagem na área gerando uma interrupção para o processador vizinho; quando o processador vizinho consome a mensagem, ele gera uma interrupção para o processador emissor indicando este consumo. A forma mais natural de implementar esta interrupção na MPH é escrever na área de recepção do processador emissor. Um campo para uma variável foi reservado em cada área compartilhada para esta finalidade.

Generalizando esta situação, sempre que existir uma mensagem na área compartilhada, e esta mensagem for consumida, a variável será sinalizada (figura III.18).

III.9 - Bloqueio Perpétuo e Espera Indefinida

Em um sistema multiprogramado, um estado de **bloqueio perpétuo** ("deadlock") existe quando um ou mais processos esperam pela ocorrência de um evento que nunca ocorrerá. Um exemplo simples de um **bloqueio perpétuo** é mostrado na alocação de recursos em um sistema com dois processos, P1 e P2, e dois recursos, R1 e R2, DEITEL (1984). Os processos precisam alocar os dois recursos para executar e, quando o fazem, só os liberam após seus términos. Portanto, o processo P1 aloca o recurso R1 e o processo P2 aloca o recurso R2. O passo seguinte para o processo P1 seria alocar o recurso R2, e para o processo P2 seria alocar o

recurso R1. Porém esses passos não podem ser executados, pois os recursos já estão alocados, ficando os processos em espera (figura III.19). Como os recursos só são liberados quando os processos terminam, e os processos estão impedidos de prosseguir, existe um estado de **bloqueio perpétuo.**

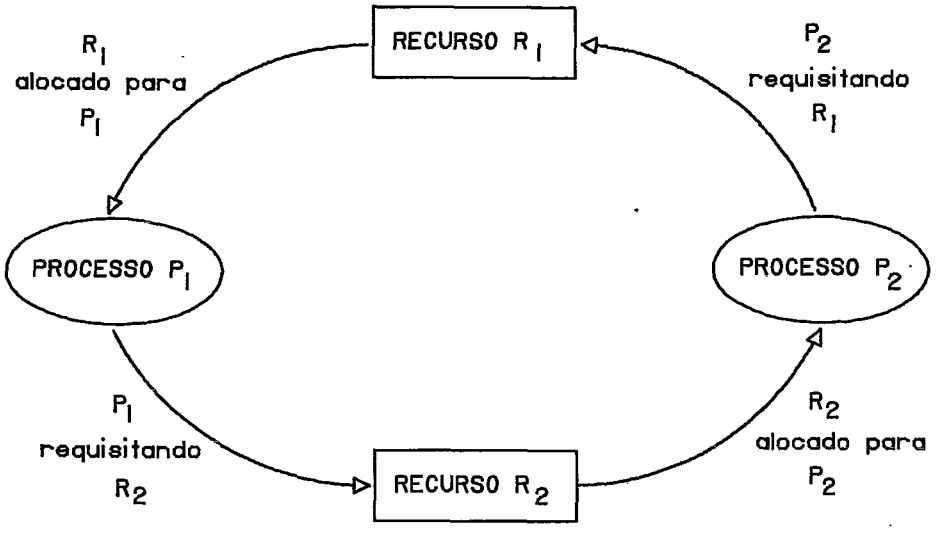


FIGURA III.19 - DIAGRAMA DE ALOCAÇÃO DE RECURSOS

Esta situação de **bloqueio** pode ocorrer também quando um processo espera pela ocorrência de um evento que é ocasionado por um segundo processo, que por sua vez também está esperando a ocorrência de um evento gerado por um terceiro processo, e assim por diante, até que o último espera pela ocorrência de um evento gerado pelo primeiro processo. Esta situação caracteriza um ciclo de espera. Quanto maior for o sistema, mais complexas podem ser as situações, pelo número de processos e recursos envolvidos

(processadores, disco, impressora, etc.). Porém, a formação desta situação, complexa ou não, é desatrosa para o sistema.

Um problema de **espera indefinida** ("indefinite postponement"), por outro lado, está diretamente relacionado à política de alocação de recursos do sistema. Por exemplo, se os recursos são alocados segundo uma política de prioridades, pode ocorrer que um processo seja adiado eternamente por sempre existir um processo com mais alta prioridade que ganhe o recurso e seja executado. Um tratamento possível para a **espera indefinida** seria aumentar a prioridade de um processo em espera com o passar do tempo, de forma que em um certo instante a sua prioridade permita a alocação do recurso necessário à sua execução.

O problema de **bloqueio perpétuo** pode ser analisado nos projetos de sistemas seguindo-se um dos seguintes métodos:

- Prevenir o **bloqueio perpétuo** garantindo que pelo menos uma das quatro condições citadas anteriormente não ocorra;
- Detectar quando o sistema entra no estado de **bloqueio** e tentar fazer que ele saia;
- Evitar o **bloqueio** através de ações preventivas apropriadas, como por exemplo, não conceder um recurso a um processo se esta ação puder causar o **bloqueio.**

Para cada um destes métodos são aplicadas várias técnicas que estão descritas em HOLT (1978), PETERSON (1985), LISTER (1984) e DEITEL (1984).

III.9.1 - Bloqueio Perpétuo na MPH

Algumas situações podem levar a MPH a um estado de **bloqueio perpétuo:**

- Comunicação entre processos - Se for alcançada uma configuração na qual todos os processos ficam bloqueados à espera de espaço para o envio de mensagens, e a liberação destes espaços depende da continuidade dos processos;
- Um erro que também pode levar o sistema ao estado de bloqueio é quando todos os processos requisitam o envio de uma mensagem que é maior que a área total disponível. Portanto, algumas restrições quanto ao tamanho de mensagens, em relação às áreas disponíveis na MPH, serão colocadas durante a implementação.

O objetivo destas restrições será impedir que o sistema alcance o estado de **bloqueio perpétuo.**

III.9.2 - Espera Indefinida na MPH

Uma situação de **Espera Indefinida** pode ocorrer devido à política de concessão dos "buffers" liberados. Um processo que ficou a espera de "buffer" para enviar uma mensagem muito grande, poderá ser adiado se somente um número pequeno de "buffers" tiver sido liberado e o processo seguinte na lista puder ser atendido. Esta situação pode se repetir indefinidamente.

CAPÍTULO IV

IMPLEMENTAÇÃO DO NÚCLEO

A implementação do Núcleo de Sistema Operacional da MPH foi realizada em quatro etapas:

- Definição das Estruturas de Dados;
- Elaboração dos Algoritmos;
- Simulação dos Algoritmos em um microcomputador IBM-PC compatível;
- Teste na MPH.

Como foi descrito na seção II.5 a MPH está interligada ao microcomputador COLOR-64. O programa do Núcleo foi desenvolvido neste microcomputador e preparado para entrar na MPH também através dele. Para esta implementação foi escolhida a linguagem PASCAL, por ser a única linguagem com compilador disponível, o DEFT PASCAL, DEFT(1984).

A etapa de simulação dos algoritmos em um microcomputdor IBM-PC compatível agilizou a implementação, pela facilidade de se encontrar ferramentas de auxílio na depuração de programas em praticamente todas as linguagens.

A listagem do código fonte do Núcleo da MPH para a etapa de simulação dos algoritmos em linguagem PASCAL é mostrado no apêndice A.

IV.1 - Definição das Estruturas de Dados

De acordo com as especificações descritas no capítulo anterior, serão apresentadas as estruturas utilizadas em cada elemento de processamento:

- Identificadores de Processo;
- **Descritores de Processo;**
- **Lista de Descritores Vazios;**
- **Lista de Processos Prontos;**
- Lista de **Processos Bloqueados pela Criação de Novos Processos;**
- Lista de **Processos Bloqueados por Temporização;**
- Lista de **Processos Bloqueados à Espera de Mensagem;**
- Lista de **Processos Bloqueados à Espera de "Buffer";**
- Lista de **Processos Bloqueados à Espera de Área Compartilhada;**
- Lista de **Mensagens Disponíveis;**
- **Tabela de Processos;**
- **Tabela de Rotas;**
- **Tabela de Rotinas;**
- Área de "Buffers";
- Área de Memória Compartilhada;
- Variáveis do Sistema.

Além dessas estruturas foram definidas algumas constantes úteis para documentação do programa e para futuras modificações. Algumas constantes definem o dimensionamento do Núcleo

- **Pdores** - Número de processadores da máquina;
- **NumProc** - Número de processos ativos permitidos;

. **NumBuf** - Número de "buffers" disponíveis;

. **TamFixo** - Tamanho da área de mensagem do "buffer";

. **TamArea** - Tamanho da área de mensagem da área compartilhada;

. **TimeSlice** - Quantidade de tempo que um processo pode ser executado sem ser interrompido;

. **NumRot** - Número máximo de rotinas que podem ser carregadas.

Outras constantes definem os estados que um processo pode assumir:

. **Executando** - Estado do processo que em um certo instante ocupa o processador;

. **Pronto** - Estado do processo que se encontra pronto para executar;

. **BloqMens** - Estado do processo bloqueado à espera de mensagem;

. **BloqTempo** - Estado do processo bloqueado por temporização;

. **BloqBuffer** - Estado do processo bloqueado à espera de buffer para enviar mensagem;

. **BloqCompar** - Estado do processo bloqueado esperando por espaço para enviar mensagem na área compartilhada;

. **BloqCriacao** - Estado do processo bloqueado por criar novos processos.

Também são utilizadas constantes para definição do tipo de erro encontrado na execução de uma **Primitiva**:

. **FaltaBuffer** - não existe "buffer" suficiente para armazenar a mensagem;

- **FaltaArea** - não existe espaço suficiente na área compartilhada para armazenar a mensagem;
- **ExcessoProc** - foi atingido o limite máximo de processos ativos em um processador;
- **ProcIne** - processo inexistente na máquina.

IV.1.1 - Identificadores de Processo

A primeira estrutura a ser apresentada é a identificação de um processo. A implementação das outras estruturas do Núcleo está intimamente ligada à estrutura de identificação dos processos. Como foi colocado na seção III.2 são considerados três parâmetros:

- processador no qual o processo existe;
- rotina cujo código o processo executa;
- instância da rotina que o processo representa.

As variáveis inteiras no DEFT PASCAL ocupam 16 bits, ou seja, uma word, e podem conter números positivos ou negativos, de acordo com o estado do bit mais significativo. Se este bit não for levado em consideração, só trataremos com valores positivos. A figura IV.1 mostra como os bits foram divididos de forma a representar o identificador de processo. Os bits 14 e 13 são preenchidos com o número do processador menos 1, variando de 0 a 3 para os processadores 1 a 4, respectivamente. Os 5 bits seguintes contém o nome da rotina, portanto são permitidas até 31 rotinas (o valor 0 não é permitido). Os últimos 8 bits determinam a intância da rotina que o processo representa, podendo variar de 1 a 254. Assim o limite de

identificadores de processos na máquina é: em cada processador são possíveis 254 instâncias de cada uma das 31 rotinas. Na atual implementação do Núcleo, no entanto, são permitidos um número menor de processos ativos por processador. Os valores 0 para rotina e para instância de rotina não estão permitidos para que nenhum processo possua o identificador com valor 0, que está reservado para o sistema.

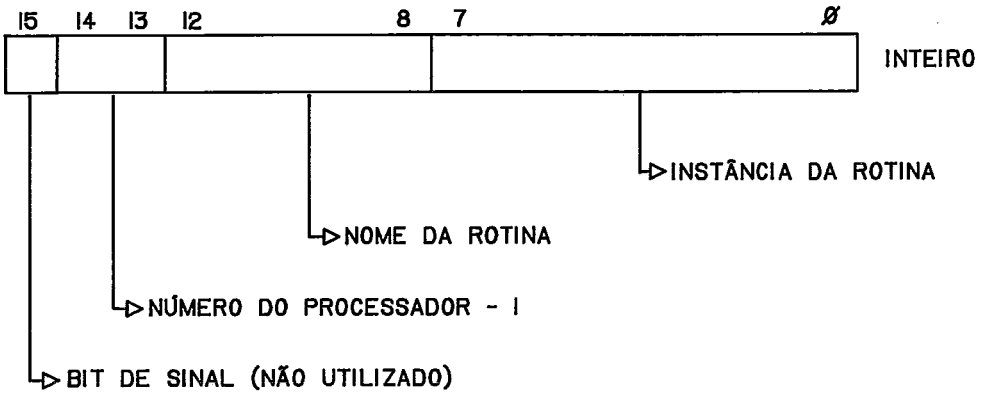


FIGURA IV.1 - FORMATO DO IDENTIFICADOR DE PROCESSO

IV.1.2 - Descritores de Processos

Os **Descritores de Processo** estão representados em uma estrutura RECORD, chamada **DescProc**, com campos para cada uma das informações apresentadas na seção III.2:

- **Proximo** - aponta para o próximo descritor;
- **NomeProc** - identificador do processo;
- **Pai** - identificador do seu processo pai;
- **Processador** - processador onde o processo executa;
- **Filhos** - quantidade de processos filhos;

- **LocalMem** - localização do código na memória;
- **LocalDado** - localização dos dados na memória;
- **LocalPilha** - localização da pilha na memória;
- **Estado** - estado do processo;
- **ProcComunica** - guarda o processo com o qual deseja se comunicar;
- **EndMensagem** - endereço da estrutura que contém ou conterá a mensagem;
- **Contexto** - guarda o contexto quando o processador é retirado da execução;
- **ListaMens** - lista de **Mensagens Disponíveis** do processo;
- **IntervEspera** - intervalo de tempo de espera;
- **FatiaTempo** - fatia de tempo ("time-slice") do processo.

A área de descritores compõe-se de uma esrutura ARRAY, com nome **ListaDesc**, contendo uma quantidade determinada pelo limite de processos ativos permitidos em cada processador, a constante **NumProc**. Durante a inicialização esses descritores são encadeados em uma lista de **Descritores Vazios** que possue uma estrutura RECORD, denominada **ListaDPont**, contendo apontadores para o início e o fim, e é manipulada como uma FIFO.

IV.1.3 - Listas do Sistema

As listas do sistema estão implementadas em estruturas RECORD de dois tipos (utilizadas de acordo com a manipulação necessária):

- **ListaDPont** - com apontadores para o início e o fim;
- **ListaPont** - com apontador somente para o início.

As listas de **Processos Prontos (ProcProntos)**, **Processos Bloqueados á Espera de "Buffer" (BloqBuf)** e **Processos Bloqueados á Espera de Área Compartilhada (BloqComp)**, onde a ordem na qual os **Descritores de Processo** são inseridos e retirados da lista é importante para a manipulação especificada, são do tipo **ListaDPont**. A lista de **Mensagens Disponíveis (ListaMens)** que faz parte do **Descritor de Processo** é implementada com o tipo **ListaDPont** pois a ordem de chegada das mensagens é levada em consideração.

As listas de **Processos Bloqueados pela Criação de Novos Processos (BloqCria)** e **Processos Bloqueados á Espera de Mensagem (BloqEMens)** são do tipo **ListaPont**, por dois motivos: não importa a ordem de inserção na lista; e, a retirada é feita percorrendo-se a lista desde o início. Na lista de **Processos Bloqueados por Temporização (BloqInterv)** a inserção de descritores é feita em qualquer posição, percorrendo-se a lista desde o início. O descritor que é retirado da lista é sempre o primeiro. Por este motivo, esta lista está implementada com o tipo **ListaPont**.

IV.1.4 - Tabelas do Sistema

A **Tabela de Processos** está implementada como uma matriz de dimensão 2, onde as linhas correspondem aos processadores e as colunas correspondem aos processos. O número de processadores da máquina está determinado pela constante **Pdores** e na atual implementação da MPH é igual a quatro (4). O número de processos é determinado pela constante **NumProc**. Esta estrutura usa uma ARRAY com dois

índices e é chamada **TabProc**.

A **Tabela de Rotas** usa somente uma estrutura ARRAY, onde os índices dos elementos são os números dos processadores e o conteúdo dos elementos é o número do processador que serve de rota. Esta tabela recebeu o nome **TabRota**. A quantidade de elementos também é determinada pela constante **NumProc**.

A **Tabela de Rotinas** é implementada com uma estrutura ARRAY, denominada **TabRotinas**, cujos elementos são também estruturas RECORD, chamadas **CelTabela**, que possuem os campos descritos na seção III.6:

- **Nome** - nome da rotina;
- **EndCarga** - endereço da memória onde a rotina deve ser carregada;
- **TamCarga** - tamanho da memória que a rotina ocupa;
- **EndPilha** - endereço da pilha da rotina;
- **TamPilha** - tamanho da pilha da rotina;
- **EndExec** - endreço inicial de execução.

A quantidade máxima de elementos que esta tabela pode conter é determinada pelo número máximo de rotinas em um processador (que é de 31) que está definida na constante **NumRot**.

IV.1.5 - Área de "Buffers"

O "Buffer" é implementado com uma estrutura RECORD, denomidada **Buffer**, contendo dois campos para apontadores,

denominados **Ponteiro** e **Cadeia** (com finalidades descritas na seção III.8.1), e uma área do tipo ARRAY com um tamanho fixo, determinado pela constante **TamFixo**, chamada **Mens**. A área de "buffers" também é uma estrutura ARRAY, denominada **AreaBuf**, e possui uma quantidade de "buffers" estipulada pela constante **NumBuf**.

A lista de **"Buffers" Livres**, que recebe o nome **BufLivre**, também é do tipo **ListaPont** e é manipulada inserindo-se e retirando-se do início (como uma pilha). Na verdade o tipo de manipulação não importa, já que quando os "buffers" são liberados eles podem ser reutilizados em qualquer tempo.

Foi especificada também uma variável chamada **QuantLivres** que possui sempre o número de "buffers" que estão didponíveis e podem ser alocados. A consulta à esta variável facilita a codificação do Núcleo, pois uma mensagem só é escrita na área de "buffers" se existirem "buffers" suficientes para guardá-la totalmente.

Durante a inicialização todos os "buffers" são alocados na lista de **"Buffers" Livres** e a variável **Quant Livres** tem como conteúdo o número total de "buffers" disponíveis.

IV.1.6 - Área de Memória Compartilhada

Cada partição dos bancos de memória compartilhada em que ocorre escrita ou leitura, está definida em uma estrutura RECORD, denominada **MemComp**, com quatro campos:

. **UltPosCheia** - apontador de última posição cheia do

"buffer" circular;

. **UltPosLivre** - apontador de última posição vazia do "buffer" circular;

. **AreaMens** - "buffer" circular;

. **MensConsumida** - indicação de mensagem consumida no processador vizinho (interrupção-para-trás).

O "buffer" circular por sua vez está implementado em uma estrutura ARRAY tendo o número de elementos limitado pela constante **TamArea**.

Cada processador, na implementação atual, necessita de duas áreas compartilhadas para enviar mensagens e mais duas para receber mensagens. O processador que fica interligado com o microcomputador COLOR-64 necessita de mais um conjunto de áreas de envio e recepção.

A manipulação desta estrutura já foi detalhada na seção III.8.2.

IV.1.7 - Variáveis do Sistema

A fim de tornar a implementação mais clara e simples, algumas variáveis também foram especificadas:

. **ProcAtual** - sempre contém o identificador do processo que está executando no momento;

. **DescAtual** - sempre contém o índice do **Descritor de Processo** que está executando no momento;

. **Processador** - número do próprio processador;

. **ParaConsumir** - indica que existe mensagem para ser

consumida na área compartilhada, ou seja, à espera de "buffer" disponível.

IV.2 - Algoritmos Detalhados

Os algoritmos detalhados nesta seção foram divididos de acordo com as estruturas que manipulam, e também de acordo com as funções que exercem em cada processador. São eles:

- Manipulação das Listas;
- Manipulação da Área de "Buffers";
- Manipulação da Área Compartilhada;
- Escalonador de Processos;
- Tratamento de Interrupção;
- **Primitivas** do Núcleo.

IV.2.1 - Manipulação das Listas

IV.2.1.1 - Rotinas Gerais

Algumas rotinas gerais foram desenvolvidas para inserção e remoção de **Descritores de Processo** das listas do Núcleo. Para as listas que possuem dois apontadores, como as listas **DescVazios** e **ProcProntos**, foram criadas as rotinas **InsLDPont** e **RetLDPont**, para inserção e remoção, respectivamente. Para as listas que possuem somente um apontador, como as listas **BloqCria** e **BloqEMens**, foram criadas as rotinas **InsLPont** e **RetLPont**, para inserção e remoção, respectivamente.

- Algoritmo da Rotina **InsLDPont**

  . se não existe descritor na lista (início=0) então

    . o descritor a ser inserido passa a ser o início

    . senão o descritor passa a ser o fim da lista

  . atualiza o fim da lista com o índice do descritor

  . zera o apontador de próximo descritor

  . atualiza o campo **Estado** do descritor

- Algoritmo da Rotina **RetLDPont**

  . se ainda existe descritor na lista (início <> 0) então

    . devolve o índice do início da lista

    . atribui ao início o índice do próximo descritor na lista

    . se não existe mais descritor na lista (início=0)

      . então o fim da lista passa a ter 0

  . senão não devolve nenhum descritor

- Algoritmo da Rotina **InsLPont**

  . se não existem mais descritores na lista (início=0)

    . então zera o apontador de próximo do descritor a ser inserido

    . senão atualiza o apontador de próximo com o valor do início

  . atualiza o início com o índice do descritor

  . atualiza o campo **Estado** do descritor

- Algoritmo da Rotina **RetLPont**

  . se o descritor não for o primeiro da lista então

    . enquanto não achar o descritor faça

. passa para o próximo descritor da lista

. atualiza os apontadores dos descritores da lista

. senão atualiza o início com o apontador de próximo do primeiro descritor da lista

. zera o apontador de próximo do descritor retirado

IV.2.1.2 - Lista de **Descritores Vazios**

Inicialmente todos os descritores estão alocados nesta lista, denominada **DescVazios**. A medida que os processos vão sendo criados, esses descritores vão sendo alocados e quando os processos terminam, os descritores são liberados voltando para a lista **DesVazios**. Os descritores são sempre inseridos no fim e retirados do início da lista. As rotinas utilizadas para estas manipulações são **InsLDPont** e **RetLDPont**, respectivamente.

IV.2.1.3 - Lista de **Processos Prontos**

Esta lista, chamada **ProcProntos** inicialmente está vazia e quando os processos são criados, seus descritores são inseridos nesta lista para que o escalonador escolha quem vai executar. A retirada de descritores ocorre quando o processo é **bloqueado**, quando o "time-slice" acaba ou quando o processo termina. Os descritores são sempre inseridos no fim e retirados do início da lista. O processo que está executando no momento é o primeiro da lista. As rotinas utilizadas para a manipulação desta lista são **InsLDPont** e **RetLDPont**.

IV.2.1.4 - Lista de **Processos Bloqueados à Espera de Mensagem**

Esta lista, com o nome **BloqEMens**, inicialmente não possui nenhum descritor. Os descritores são sempre incluídos no início. Um descritor só é retirado quando o processo correspondente recebe a mensagem desejada. Neste momento a lista é percorrida desde o início para se encontrar o descritor e quando o descritor é encontrado ele é retirado da lista. Esta manipulação supõe que o **Descritor do Processo** já foi identificado e já foi verificado se está bloqueado. Como essa remoção de descritores é uma ocorrência aleatória, não adianta inserir os descritores na lista segundo uma ordem pré-definida. No momento em que a mensagem que um processo esperava chegou, ele deve ser liberado, não obstante, o lugar que ocupe na lista. As rotinas que executam a manipulação desta lista são **InsLPont** e **RetLPont**.

IV.2.1.5 - Lista de **Processos Bloqueados pela Criação de Novos Processos**

Inicialmente esta lista, com nome de **BloqCria**, está vazia. Quando um processo cria outros processos então o seu descritor é inserido nesta lista. O processo somente é desbloqueado quando todos os processos criados terminam. Neste momento, o descritor é retirado da lista não obstante o lugar em que se encontra. Percebe-se portanto que a ordem de inserção na lista não importa, pois os procesos são retirados em qualquer ordem. Os descritores são sempre

inseridos no início da lista, e ela é percorrida desde o início para a remoção. As rotinas que executam estas manipulações são **InsLPont** e **RetLPont**.

IV.2.1.6 - Lista de **Processos Bloqueados por Temporização**

Inicialmente esta lista, que possui o nome **BloqInterv**, está vazia. Os **Descritores de Processo** são incluídos obedecendo-se uma ordem crescente quanto ao intervalo de espera desejado. O descritor retirado é sempre o primeiro da lista. A lógica de manipulação desta lista já foi comentada na seção III.7.

- Algoritmo de Inserção de Descritor
  . se não exite descritor na lista (inicio=0) então
    . atualiza o início como índice deste descritor
    . zera o conteúdo do apontador de próximo
  . senão
    . enquanto existir descritor na lista faça
      . se o intervalo de espera do descritor a ser inserido é menor que o intervalo de espera do descritor na lista então
        . coloca o descritor na frente do descritor da lista
        . diminui o intervalo de espera do descritor da lista do valor do intervalo de espera do descritor inserido
        . se o descritor da lista era o primeiro
          . então atualiza o início com o índice do descritor inserido

. senão atualiza o apontador de próximo do descritor inserido com o índice do descritor da lista

. sai deste "loop"

. senão faça

. diminui o intervalo de espera do descritor a ser inserido do valor do intervalo de espera do decritor da lista

. se ainda existir descritor na lista então

. passa para o próximo decritor

. senão coloca o descritor no fim da lista e sai deste "loop"

. atualiza o estado do processo para **BloqTempo**

- Algoritmo de Remoção de Descritor

. fornece o primeiro descritor da lista

. atualiza o início da lista com o próximo descritor

IV.2.1.7 - Lista de **Mensagens Disponíveis**

Esta lista, chamada **ListaMens**, difere das outras pois não guarda descritores e sim "buffers", e inicialmente também está vazia. As mensagens são guardadas para os processos na ordem em que elas chegam, ou seja, são sempre inseridas no fim da lista. Na verdade somente o primeiro "buffer" da mensagem é inserido na lista . Os outros "buffers" (quando houver) estão encadeados no primeiro "buffer". A retirada destas mensagens é aleatória e só depende de qual processo origem se deseja receber mensagem. A rotina de remoção procura pela primeira

mensagem do processo origem especificado, devolvendo o índice do "buffer" se encontrá-lo ou 0 se não encontrá-lo. Uma variação da retirada de "buffers" da lista é necessária quando o processo termina e algumas mensagens ficaram pendentes. Para isto, quando a rotina de remoção for ativada com o parâmetro origem igual a 0, quer dizer que o primeiro "buffer" que estiver na lista deve ser devolvido. Para se retirar todos os "buffers" basta que a rotina de remoção seja ativada novamente enquanto não for devolvido "buffer" igual a 0.

- Algoritmo de Inserção de "Buffer"
  - . zera o apontador de próximo do "buffer"
  - . se a lista está vazia (fim = 0)
    - . então atualiza o início com o índice do "buffer"
    - . senão atualiza o apontador de próximo do último "buffer" da lista com o índice do "buffer"
  - . atualiza o fim da lista com o índice do "buffer"

- Algoritmo de Retirada de "Buffer"
  - . se origem = 0 então
    - . fornece o primeiro "buffer" da lista
    - . atualiza o início com o próximo "buffer" da lista
    - . se não existem mais "buffers" na lista (inicio=0)
      - . então o fim passa a ter 0
  - . senão faz
    - . enquanto não acabar a lista e o processo origem da mensagem não for o desejado faça
      - . aponta para o próximo"buffer" da lista
    - . se achou o "buffer" com a mensagem desejada então

. fornece o índice do "buffer"
. se o "buffer" desejado é o primeiro da lista
. então atualiza o início com o índice do próximo "buffer" da lista
. senão atualiza os apontadores dos "buffers" da lista
. se o "buffer" desejado é o último da lista
. então atualiza o fim da lista com o índice do "buffer" anterior
. se não existirem mais "buffers" na lista (inicio=0)
. então o fim passa a ter o valor 0
. senão devolve o valor 0

IV.2.1.8 - Lista de **Processos Bloqueados à Espera de "Buffer"**

No início da execução da máquina esta lista, denominada **BloqBuf**, está vazia. Os **Descritores de Processo** são inseridos nesta lista quando não há mais "buffers" para armazenar uma mensagem cujo destino é um processo do mesmo processador que o emitente. A inserção deve ser sempre no fim da lista para que a ordem de chegada das mensagens seja preservada. A retirada de descritores da lista ocorre quando um ou mais "buffers" são liberados. A rotina de retirada manipula a lista em relção a um descritor específico, pois assume-se que previamente já foi determinado que o descritor em questão pode ser liberado. A remoção, portanto, se faz em qualquer lugar da lista.

- Algoritmo de Inserção de Descritor
  . se não existe descritor na lista (inicio=0)
    . então atualiza o início com o índice do descritor
    . senão atualiza o apontador de próximo do último descritor da lista
  . atualiza o fim com o índice do descritor
  . atualiza o estado do processo para **BloqBuffer**

- Algoritmo de Remoção de Descritor
  . se o descritor a ser retirado é o primeiro da lista então
    . atualiza o início com o apontador de próximo do primeiro da lista
    . se não existe mais descritor na lista (inicio=0)
      . então o fim passa a ter 0
  . senão
    . enquanto não achar o descritor na lista faça
      . aponta para o próximo descritor na lista
    . atualiza os apontadores dos descritores da lista
    . se o descritor é o último da lista então
      . atualiza o fim com o índice do descritor anterior

IV.2.1.9 - Lista de **Processos Bloqueados à Espera de Área Compartilhada**

Inicialmente esta lista, que possui o nome **BloqComp**, não possue nenhum descritor. Quando um processo deseja enviar uma mensagem para outro processo que não se encontra no mesmo processador e não existe espaço suficiente na área

compartilhada para a mensagem, então o descritor do processo é inserido nesta lista. Os descritores são sempre inseridos no fim da lista para que a ordem seja mantida. A retirada só é realizada quando algum espaço na área compartilhada é liberado. A rotina manipula a lista em relação a um descritor específico, quer dizer, assume-se que a lista já foi percorrida definindo-se qual o descritor que podia utilizar o espaço liberado. Protanto, a remoção é feita em qualquer lugar da lista.

Os algoritmos de inserção e remoção de descritor neta lista são idênticos aos algoritmos da lista de **Processos Bloqueados à Espera de "Buffer"**, a menos do estado do processo que é atualizado para **BloqComp**.

IV.2.2 - Manipulação da Área de "Buffers"

Foram especificadas quatro rotinas para manipulação dos "buffers" desta área. São elas:

- aloca um "buffer" da lista de **BufLivre**;
- devolve um "buffer" para a lista de **BufLivre**;
- guarda em "buffers" uma mensagem com determinado endereço;
- verifica a existência de mensagens e processos esperando por liberação de "buffers".

Uma explicação mais detalhada em relação ao gerenciamento desta área pode ser encontrada na seção III.8.1.

IV.2.2.1 - Alocação de "Buffer"

A rotina **AlocaBuf** retira um "buffer" da lista **BufLivre**, devolvendo o seu índice. Os "buffers são sempre retirados do início da lista.

- Algoritmo:
  . devolve o início da lista
  . atualiza o início com o campo **Ponteiro** do "buffer" retirado
  . decrementa o número de "buffers" livres, a variável **QuantLivres**

IV.2.2.2 - Devolução de "Buffer"

Esta rotina, chamada **DevolveBuf**, é também muito simples. Os "buffers" são devolvidos sempre que a mensagem que continham foi recebida. A inserção de "buffers" também é feita no início da lista.

- Algoritmo:
  . atualiza o campo **Ponteiro** do buffer com o valor do início
  . atualiza o início da lista com o índice do "buffer"
  . incrementa a variável **QuantLivres**

IV.2.2.3 - Guarda Mensagem em "Buffers"

A rotina **GuardaMens** tem a função de guardar uma mensagem em um ou mais "buffers". Esta mensagem pode estar

em um endereço da área de dados do processo ou pode estar na área de memória compartilhada. Os parâmetros passados na ativação desta rotina são:

- identificador do processo destino da mensagem;
- identificador do processo origem da mensagem;
- endereço da mensagem (endereço de uma estrutura ARRAY);
- indicação de que a mensagem vem da área compartilhada ou da área do processo;
- a posição no ARRAY na qual a mensagem começa;
- variável para devolver o primeiro "buffer" da mensagem;
- variável para devolver a indicação de erro.

A indicação de que a mensagem veio da área comparitilhada ou da área do processo é necessária para: a interpretação dos parâmetros do endereço da mensagem e posição no ARRAY; e, o tipo de cópia em "buffer" que deve ser realizada. Quando a mensagem vem da área compartilhada, a rotina recebe o endereço do "buffer" circular (**AreaMens**) e a posição no ARRAY é o apontador de última posição vazia do "buffer" circular (**UltPosLivre**). Quando a mensagem está na área do processo, a rotina recebe o endereço da mensagem efetivamente e a posição é 0, ou seja, logo a primeira posição já é o tamanho da mensagem. Quanto ao tipo de cópia, se a mensagem está na área do processo ela ainda deve ser montada dentro do "buffer" com o cabeçalho, de acordo com o formato mostrado na figura III.8. Se por outro lado a mensagem está na área compartilhada deve-se fazer um controle da leitura do "buffer" circular.

O índice do primeiro "buffer" da mensagem é devolvido

para que depois o "buffer" possa ser inserido na lista **ListaMens** do processo destino.

- Algoritmo:
  - . se a mensagem está na área compartilhada
    - . então o tamanho da mensagem é o conteúdo do endereço indicado pelo parâmetro posição
    - . senão o tamnho da mensagem está na primeira posição da mensagem
  - . adiciona ao tamanho a quantidade de dados do cabeçalho
  - . calcula a quantidade de "buffers" necessários para armazenar a mensagem
  - . se a quantidade de "buffers" livres é suficiente então
    - . enquanto não armazenar toda a mensagem faça
      - . aloca um "buffer" da lista de **BufLivre**
      - . se é o primeiro "buffer" da mensagem então
        - . guarda o índice deste "buffer"
        - . preenche o cabeçalho no "buffer"
        - . calcula a quantidade de dados que ainda podem ser copiados neste "buffer"
        - . se a mensagem vem da área compartilhada
          - . então copia a quantidade de dados calculada, porém fazendo o controle de leitura do "buffer" circular
          - . senão copia a quantidade de dados calculada
      - . senão
        - . encadeia o "buffer" anterior com o

"buffer" atual

. calcula a quantidade de dados que podem ser copiados no "buffer"

. se a mensagem vem da área compartilhada

. então copia a quantidade de dados calculada, porém fazendo o controle de leitura do "buffer" circular

. senão copia a quantidade de dados calculada

. fim do enquanto

. senão indica erro de falta de "buffer"

IV.2.2.4 - Verifica Bloqueio por "Buffer"

A rotina **VBloqBuf** executa duas funções:

. verifica se existe mensagem na área compartilhada esperando por liberação de "buffer" para ser armazenada;

. verifica se existem processos na lista **BloqBuf**.

Quando alguns "buffers" são liberados, a rotina pesquisa as duas possibilidades descritas acima e a que puder ser satisfeita primeiro é atendida. A explicação a respeito deste tipo de escolha está analisada na seção III.8.3.

- Algoritmo:

. se existe mensagem na área compartilhada esperando "buffer" então

. enquanto a área contiver mensagem e não ocorrer

nenhum erro faça

. obtém tamanho da mensagem

. tenta guardar a mensagem em "buffers", obtendo o primeiro "buffer"

. se houve "buffer" suficiente então

. enfileira a mensagem na lista **ListaMens** do processo destino da mensagem

. atualiza o apontador **UltPosLivre** da área

. se liberou alguma área compartilhada

. então verifica se pode guardar mensagem pendente na área compartilhada

. se existe processo bloqueado esperando por "buffer" e ainda existem "buffers" disponíveis então

. obtém o primeiro descritor da lista

. enquanto existirem processos na lista **BloqBuf** e não ocorrer erro faça

. tenta guardar a mensagem em "buffers", obtendo o primeiro "buffer"

. se houve "buffer" suficiente

então faça

. enfileira a mensagem na lista **ListaMens** do processo destino da mensagem

. retira o descritor da lista **BloqBuf**

. insere o descritor na lista **ProcProntos**

senão tenta para o próximo descritor na lista

IV.2.3 - Manipulação da Área Compartilhada

As mensagens são inseridas na área de memória compartilhada quando um processo requisita o envio de uma mensagem. As mensagens são retiradas desta área pela rotina que trata a interrupção gerada pela escrita na área compartilhada. Nesta seção será detalhada a rotina de inserção de mensagens na área compartilhada, chamada **InsereMens**. A rotina de remoção será detalhada na seção seguinte que mostra as rotinas de tratamento de interrupções. Outra royina que direciona a manipulação da área compartilhada é a **VBloqComp** que verifica o bloqueio na área compartilhada também será comentada adiante.

IV.2.3.1 - Inserção na Área Compartilhada

A rotina **InsereMens** utiliza o algoritmo de manipulação da área compartilhada demonstrado no **Apêndice B**. Cada processador possui duas áreas distintas nas quais pode escrever, dependendo de qual processador vizinho queira atingir. Esta mensagem pode receber uma mensagem comum de processo ou a mensagem de destino 0 do sistema. A diferença destas mensagens é quanto ao tamanho. Se a mensagem é comum o tamanho está especificado no primeiro dado da mensagem. Se, por outro lado, é uma mensagem de sistema o tamanho é passado como parâmetro e o endereço da mensagem só contém dados da mensagem. As mensagens inseridas nesta área também seguem o formato mostrado na figura III.8. Os parâmetros que esta rotina deve receber para promover a inserção são:

. identificador do processo destino da mensagem;

. identficador do processo origem da mensagem;
. endereço da mensagem (contém o tamanho se for mensagem comum);
. tamanho da mensagem (se for mensagem do sistema);
. endereço da área (**AreaMens**) na qual a mensagem deve ser colocada (depende do processo destino);
. endereço do apontador da última posição cheia desta área (**UltPosCheia**) que será modificada durante a inserção;
. apontador da última posição vazia desta área (**UltPosLivre**);
. indicação de erro se a área não possuir espaço suficiente para a mensagem.

- Algoritmo:
  . se a área está vazia (**UltPosLivre** = **UltPosCheia**)
    . então o espaço livre é igual ao tamanho do "buffer" circular, **TamArea**
    . senão calcula o espaço livre
  . se o espaço livre é suficiente para a mensagem (tamanho + cabeçalho) então
    . se a mensagem é de sistema
      . então copia o cabeçalho da mensagem (identificador de destino e de origem)
      . senão a mensagem está completa
    . copia a mensagem na área fazendo-se o controle de escrita no "buffer" circular
    . atualiza o apontador **UltPosCheia**
  . senão indica erro de falta de espaço na área compartilhada

IV.2.3.2 - Verificação de Bloqueio por Área Compartilhada

A rotina **VBloqComp**, verifica se existem processos bloqueados à espera de liberação de espaço na área compartilhada. Se existir espaço, os processos vão sendo retirados da lista e desbloqueados, e suas mensagens vão sendo escritas na área compartilhada até que não haja mais espaço. O único parâmetro que é passado para esta rotina é a indicação de em qual área compartilhada houve a liberação de espaço.

- Algoritmo:
  . enquanto existir processo na lista e não ocorrer erro faça
    . tenta inserir a mensagem na área compartilhada correspondente
    . se houve espaço suficiente então
      . retira o processo da lista **BloqComp**
      . insere o processo na lista **ProcProntos**

IV.2.4 - Escalonador de Processos

O escalonamento de processos na MPH está descrito na seção III.4. A rotina **Escalonador** é ativada sempre que um processo sai do início da lista de **Processos Prontos** (**ProcProntos**). De acordo com a política especificada, o primeiro descritor da lista ganha o processador nesta situação e se o campo **FatiaTempo** estiver com o valor 0, então recebe um "time-slice" para executar. Pode-se alcançar uma situação em que nenhum processo ativo esteja

na lista **ProcProntos** esperando para executar. Quando isto ocorre o escalonador deve ativar um processo ocioso do Sistema Operacional que não possue fatia de tempo para executar. No entanto, este processo será interrompido assim que um processo for inserido na lista **Proc Prontos.**

- Algoritmo:
  . se não existem descritores na lista **ProcProntos** (inicio=0) então
    . ativa processo ocioso do sistema
  . senão
    . atualiza o estado do primeiro processo da lista **ProcProntos** para **Executando**
    . atualiza a variável **ProcAtual** com o identificador deste processo
    . atualiza a variável **DescAtual** com o índice deste descritor
    . se o campo **FatiaTempo** está com 0 então
      . atualiza o **FatiaTempo** com o valor do "time-slice"

IV.2.5 - Tratamento de Interrupção

Como foi colocado na seção III.1, somente dois tipos de interrupção ocorrem na MPH para serem tratadas pelo Núcleo. São elas:

. "tick" do relógio de tempo real;
. indicação de escrita na área compartilhada, ou seja, recepção de mensagem.

IV.2.5.1 - Interrupção de Relógio

É uma interrupção gerada periódicamente, indicando que um "tick" do relógio se passou. Esta interrupção é utilizada por duas funções do Núcleo e foram detalhadas na seção III.7. A cada "tick" do relógio o campo **FatiaTempo** do **Descritor do Processo** que está executando deve ser decrementado. Se este campo chegar a zero (acabou o "time-slice") o descritor deve ser retirado da lista **ProcProntos** e o escalonador deve ser ativado.

Se existir algum processo bloqueado por temporizaçao (lista **BloqInterv**) o campo **IntervEspera** do primeiro descritor da lista é decrementado. Quando este campo do descritor chegar a zero (0), o processo deve ser retirado da lista. Se na lista **BloqInterv** um descritor com o campo **IntervEspera** igual a 0 passar a ser o primeiro, ele deve ser retirado da lista imediatamente.

- Algoritmo:
  . se existe processo executando
    . então decrementa o campo **FatiaTempo** do primeiro descritor da lista **ProcPontos**
    . se este campo do descritor chegou a zero então
      . retira este descritor do início da lista
      . insere o descritor no fim da lista
      . ativa o escalonador de processos
  . se existe algum processo na lista **BloqInterv** (inicio<>0) então
    . decrementa o campo **IntervEspera** do primeiro da

lista

. enquanto existirem descritores na lista com campo **IntervEpera** igual a 0 faça

. retira o descritor da lista **BloqInterv**

. insere o descritor na lista **ProcProntos**

. se não existe processo ativo executando

. então ativa o escalonador de processos

IV.2.5.2 - Interrupção de Recepção de Mensagem

Esta rotina, chamada **IntRx**, é ativada quando uma interrupção de recepção ocorre. Cada processador pode receber mensagem de duas áreas compartilhadas. Um "flip-flop" mapeado em um endereço da memória do processador contém a informação de qual processador gerou a interrupção. Um primeiro nível de tratamento, portanto, lê este endereço e direciona uma rotina de consumo de mensagens para uma área específica. Este primeiro nível também verifica se a interrupção foi acionada apenas para indicar um consumo de mensagem (interrupção-para-trás).

- Algoritmo:

. lê o conteúdo do "flip-flop"

. caso este conteúdo seja (no caso do processador N)

. N+1 faça

. se foi indicação de consumo de mensagem

. então foi liberado algum espaço na área compartilhada

. senão consome a mensagem da área compartilhada com o processador N+1

. N-1 faça

. se foi indicação de consumo de mensagem

. então foi liberado algum espaço na área compartilhada

. senão consome a mensagem da área compartilhada com o processador N-1

A rotina **ConsomeMensagem**, que consome as mensagens da área compartilhada posuui três funções:

. tratar as mensagens de sistema (destino=0) e/ou roteá-las para outro processador;

. tratar as mensagens comuns destinadas a processos residentes no processador;

. rotear as mensagens comuns destinadas a processos residentes em outros processadores;

No momento em que esta rotina é ativada todas as mensagens que estiverem na área compartilhada devem ser tratadas a não ser que ocorra algum impedimento como:

. "buffers" suficientes para armazenar a mensagem não estão disponíveis;

. a área para onde a mensagem deve ser roteada está cheia.

- Algoritmo:

. enquanto a área compartilhada não estiver vazia e não ocorrer erro faça

. lê o destino da mensagem e alguns parâmetros

. se a mensagem é de sistema (destino=0) então

. caso o tipo da mensagem seja

. carregamento de rotina faça
    . se é para este processador então
        . guarda as informações na **TabRotinas** do processador
        . carrega a rotina na memória
    . senão ativa o roteamento de mensagem
. criação de processo faça
    . se é para este processador então
        . cria um novo processo (utiliza a **Primitiva** do Núcleo)
    . senão ativa o roteamento de mensagem
. inclusão de processo na **TabProc** dos processadores faça
    . se não foi o próprio processador que gerou a mensagem então
        . procura um espaço vazio na **TabProc** e guarda o identificador
        . ativa o roteamento de mensagem
. término de um processo (mensagem de difusão) faça
    . se não foi o próprio processador que gerou a mensagem então
        . procura a posição na **TabProc** na qual existia este processo limpando-a
        . se o processo pai resiste neste processador então
            . avisa que um de seus processos filhos terminou
        . ativa o roteamento de mensagem

. libera o espaço da mensagem
. senão (mensagem comum)
  . se o processo destino desta mensagem está neste processador então
    . se o processo destino está bloqueado à espera de mensagem então
      . se o processo destino está esperando mensagem deste processo origem então
        . copia a mensagem para o endereço especificado no descritor do processo destino
        . retira o descritor da lista **BloqEMens**
        . insere o descritor na lista **ProcProntos**
    . guarda a mensagem na área de "buffers"
    . se não ocorreu erro então
      . insere a mensagem na lista **ListaMens** do processo destino
      . senão indica que existe mensagem pendente na área compartilhada
    . se não ocorreu erro então
      . libera a área compartilhada
  . senão roteia a mensagem para o processador correto

O roteamento de mensagens é executado por uma rotina de nome **Roteia**, que somente possui a função de procurar a rota correta e ativar a rotina de inserção na área compartilhada. Esta rotina tem como parâmetros:

. processador destino da mensagem;
. identificador do processo destino;
. identificador do processo origem (ou tipo se for mensagem de sistema);
. endereço da mensagem;
. tamanho da mensagem;
. indicação de ocorrência de erro.

- Algoritmo
  . escolhe a rota adequada
  . caso a rota seja (no caso do processador N)
    . N+1 faça
      . insere a mensagem na área compartilhada com o processador N+1
    . N-1 faça
      . insere a mensagem na área compartilhada com o processador N-1

IV.2.6 - **Primitivas** do Núcleo

O Núcleo oferece aos processos os serviços básicos para utilização dos recursos da máquina. As **Primitivas**, que são as rotinas do Sistema Operacional, executam esses serviços. Para a ativação das **Primitivas** são necessários alguns parâmetros que serão analisados adiante. As **Primitivas** deste sistema são:

. **Envia** - envia mensagens de um processo para outro;
. **Recebe** - fornece a mensagem enviada por um processo específico;
. **Espera** - espera por um intervalo de tempo determinado

sem execução;

. **CriaProc** - cria processos filhos de um processo;

. **Termina** - avisa o término de um processo;

IV.2.6.1 - Envia Mensagem

A **Primitiva Envia** executa o serviço de envio de uma mensagem de um processo para outro. As características e o funcionamento desta **Primitiva** estão descritos na seção III.8. Os parâmetros que devem ser passados para esta rotina são:

. o processo destino da mensagem, que é identificado por:

. número do processador;

. nome da rotina;

. instância da rotina.

. endereço da estrutura ARRAY que contem o tamanho e a mensagem;

Se o processo destino da mensagem já estiver **bloqueado** à espera de uma mensagem deste processo origem, ele deve ser desbloqueado e inserido na lista **ProcProntos**. Se a mensagem não puder ser enviada porque falta espaço ("buffer" ou área compartilhada), o processo deve ser **bloqueado** até que o espaço necessário para sua mensagem seja liberado.

- Algoritmo:

. verifica se o processo existe na **TabProc**

. se existir então

. se o processo destino está no mesmo processador

então

. descobre o descritor do processo destino

. se o processo destino está bloqueado à espera de mensagem então

. se a mensagem que ele espera é deste processo origem então

. copia a mensagem para o endereço especificado no descritor do processo origem

. retira o processo destino da lista **BloqEMens**

. insere na lista **ProcProntos**

. senão

. guarda a mensagem nos "buffers"

. se existiam "buffers" suficientes

. então insere o primeiro "buffer" na lista de **ListaMens** do processo destino

. senão

. retira este processo da lista **ProcProntos**

. insere este processo na lista **BloqBuf**

. guarda no descritor o identificador do processo destino e o endereço da estrutura da mensagem

. ativa o escalonador de processos

. senão

. coloca a mensagem na área compartilhada com o

processador onde existe o processo destino ou na área compartilhada com o processador que serve de rota

. se não houve espaço suficiente então
  . retira este processo da lista **ProcProntos**
  . insere este processo na lista **BloqComp**
  . guarda no descritor o identificador do processo destino e o endereço da estrutura da mensagem
  . ativa o escalonador de processos

. senão devolve um erro indicando que o processo destino não existe

IV.2.6.2 - Recebe Mensagem

O recebimento de mensagem também está comentado na seção III.8. A **Primitiva Recebe** quando é ativada procura por uma mensagem de um processo origem específico na lista **ListaMens** e se não encontra, o processo requisitante é **bloqueado**. Os parâmetros desta rotina são:

. o número do processador do processo origem;
. o nome da rotina do processo origem;
. a instância da rotina que o processo origem representa;
. o endereço da estrutura onde a mensagem deve ser guardada.

As mensagens que estão na lista **ListaMens** estão armazenadas em um ou mais "buffers". Quando esses "buffers" são liberados esta rotina automaticamente procura por processos que estavam bloqueados por falta de "buffers".

- Algoritmo:
  . se não existe nenhuma mensagem na lista do processo (inicio=0) então
    . retira este processo da lista **ProcProntos**
    . insere este processo na lista **BloqEMens**
    . guarda no descritor o identificador do processo origem e o endereço da estrutura que deverá conter a mensagem recebida
    . ativa o escalonador de processos
  . senão
    . procura na lista **ListaMens** se existe a mensagem desejada, obtendo o primeiro "buffer"
    . se existir então
      . obtém o tamanho da mensagem
      . se a mensagem ocupa mais de um "buffer" então
        . copia para a estrutura especificada o conteúdo do primeiro "buffer"
        . calcula a quantidade de dados da mensagem que ainda devem ser copiados
        . enquanto não acabarem os "buffers" com a mensagem faça
          . calcula a quantidade de dados do "buffer" que ainda devem ser copiados
          . obtém o próximo "buffer"
          . devolve o "buffer" anterior para a lista de **BufLivre**
          . copia para a estrutura especificada mais uma parte da mensagem
        . devolve o último "buffer"
      . senão

. copia a mensagem para a estrutura especificada

. devolve o "buffer"

. se existem mensagens pendentes para serem guardadas em "buffers" então tenta guardá-las

. senão

. retira este processo da lista **ProcProntos**

. insere este processo na lista **BloqEMens**

. guarda no descritor o identificador do processo origem e o endereço da estrutura que deverá conter a mensagem recebida

. ativa o escalonador de processos

IV.2.6.3 - Espera Intervalo

Esta **Primitiva** apenas retira o processo que está executando da lista **ProcProntos, bloqueando-o** pelo tempo desejado. O escalonador é ativado para que um novo processo entre em execução. O único parâmetro necessário para esta rotina é a quantidade de tempo que o proceso deseja esperar.

- Algoritmo:

. guarda no campo **InterEspera** descritor do processo, a quantidade de tempo desejada

. retira este processo da lista **ProcProntos**

. insere este processo na lista **BloqInterv**

. ativa o escalonador de processo

IV.2.6.4 - Criação de Processo

Um processo pode criar tantos processos quantos quiser, respeitando o limite máximo de processos ativos em um processador. Processos podem ser criados no mesmo processador ou em processadores diferentes. Os parâmetros necessários à rotina de criação de processos, chamada **CriaProc**, são:

- quantidade de processos a serem criados;
- indentificador do processo pai (é igual a 0 quando o Núcleo ativa os primeiros processos);
- endereço de uma estrutura que contém a informação dos processos a serem criados;
- variável para guardar a indicação de erro quando o limite máximo de processos ativos for atingido.

A estrutura que contém as informações para crição de processos é implementada por um ARRAY e possui os seguintes campos:

- número do processador;
- nome da rotina;
- quantidade de instâncias da rotina.

Se um processo desejar criar vários processos, em processadores distintos e com rotinas diferentes, basta adicionar na mesma estrutura um conjunto desses três campos para cada rotina e/ou processador diferente. Naturalmente o parâmetro de quantidade de processos a serem criados desta **Primitiva**, deverá, neste caso, conter a quantidade total de processos que serão criados em toda a máquina.

Quando os processos são criados no próprio processador, um descritor deve ser alocado e preenchido com as informações necessárias, a **TabProc** do procesador é atualizada e é gerada uma mensagem de inclusão de processo nas **TabProc's** dos outros processadores.

Quando um processo pede a criação de novos processos, ele deve ser **bloqueado** e esperar até que todos os seus processos filhos acabem. Quando o Núcleo ativa esta rotina para gerar os primeiros processos, não existe este bloqueio.

- Algoritmo:
  . enquanto não forem criados todos os processos faça
    . aponta para um conjunto de três campos da estrutura
    . se é para criar processos em outro processador então
      . envia mensagem de sistema para criação de processos em outro procesador
      . diminui estes processos da quantidade total de processos
    . senão
      . enquanto houverem instâncias a serem criadas e não ocorreu nenhum erro
        . aloca um descritor da lista **DescVazios**
        . se conseguiu alocar (existia descritor) então
          . diminui o número de instâncias
          . diminui a quantidade total de

processos
. preenche o descritor
. atualiza a **TabProc** deste processador
. insere o descritor na lista **ProcProntos**
. envia mensagem de inclusão de processo na **TabProc** dos outros processadores
. senão indica que ocorreu um erro por ter excedido o limite máximo de processos ativos
. se não ocorreu nenhum erro então
. se é um processo comum que cria os processos e ele reside neste processador então
. guarda a quantidade total de processos criados no descritor do processo
. retira este processo da lista **ProcProntos**
. insere este processo na lista **BloqCria**
. ativa o escalonador de processos

IV.2.6.5 - Término de Processo

Todo o processo que termina na máquina deve requisitar a **Primitiva Termina** que desativa o processo. Esta rotina tem por função:

. avisar ao processo pai deste processo que um de seus processos filhos acabou;
. retirar este processo da **TabProc** deste processador;
. descartar todas as mensagens que ficaram pendentes neste processo;

. avisar aos outros processadores da máquina desta ocorrência.

Esta rotina não possui parâmetros já que o processo que ativa esta **Primitiva** está executando no momento e todas as informações para seu término estão contidas no seu descritor.

- Algoritmo:
  . envia uma mensagem a todos os processadores e também ao seu processo pai avisando o término
  . procura na **TabProc** a identificação do processo, limpando-a
  . enquanto existirem mensagens pendentes na lista **ListaMens** do processo faça
    . retira o "buffer" da lista **ListaMens**
    . devolve o "buffer" para a lista **BufLivre**
    . enquanto existirem "buffers" encadeados faça
      . obtém o "buffer"
      . devolve o "buffer" para a lista **BufLivre**
  . se o pai deste processo se encontra neste processador
    . então avisa o seu término
  . retira o descritor deste processo da lista **ProcProntos**
  . insere o descritor deste processo na lista **DescVazios**
  . ativa o escalonador de processos

IV.3 - Simulação dos Algoritmos

Para esta fase da implementação foi necessária a criação de várias estruturas que simulassem o funcionamento da máquina. Essas estruturas e sua utilização serão detalhadas nesta seção.

Na simulaçao assumiu-se que o processador corrente é o processador 1. Portanto foram criadas as áreas compartilhadas nas quais o processador 1 se comunica com os procesadores 2 e 4, chamadas **Envia1_2** e **Envia1_4**. Foi criada também a área compartilhada entre o processador 2 e o processador 3, chamada **Envia2_3**, para que fosse possível testar o roteamento de uma mensagem que vai do processador 1 para o processador 3.

Em cada processador existe também a ocorrência das interrupções que foram simuladas através das variáveis Controle_1 a Controle_4. Essas variáveis indicam a ocorrência dos dois tipos de interrupção que o Núcleo trata, quando bits específicos estão com o valor 1:

- interrupção de escrita em área compartilhada - bit 0;
- interrupção de relógio - bit 1.

Foram simulados também os "flip-flops" dos processadores que através de uma combinação de bits informam qual processador provocou a interrupção de escrita na área compartilhada. A simulação dos "flip-flops" é realizada pelas variáveis **Processador_1** a **Processador_4**.

Para controlar essas estruturas de simulação, uma rotina, chamada **Maquina**, foi codificada para direcionar o tratamento das interrupções.

- Algoritmo da rotina **Maquina**:
  - . ativa a rotina de tratamento de "tick" do relógio
  - . se o bit 0 da variável **Controle_1** está em 1
    - . então ativa a rotina de tratamento de recepção do processador 1
  - . se o bit 0 da variável **Controle_2** está em 1
    - . então ativa a rotina de tratamento de recepção do processador 2
  - . se o bit 0 da variável **Controle_3** está em 1
    - . então ativa a rotina de tratamento de recepção do processador 3
  - . se o bit 0 da variável **Controle_4** está em 1
    - . então ativa a rotina de tratamento de recepção do processador 4

A rotina **Maquina** é ativada após a execução de qualquer **Primitiva**. Desta forma, a interrupção de "tick" de relógio será sempre tratada.

Para implementar a interrupção de recepção de mensagem, basta que quando se fizer alguma escrita na área compartilhada, as variáveis **Controle** e **Processador** do processador correspondente sejam alteradas. Os exemplos destas manipulações são:

. se o processador 1 escreve na área **Envia1_2**, a variável **Controle_2** tem o bit 0 setado e a variável

Processador_2 passa a conter o valor 1. Com isso a rotina **IntRx2** é ativada e de acordo com o conteúdo da variável **Processador_2** a rotina **ConsomeMensagem_2** é direcionada a pesquisar a área **Envia1_2**, que é onde a mensagem foi escrita. No fim da rotina **IntRx2** a variável **Controle_2** tem o bit 0 clareado;

. se o processador 1 escreve na área **Envia1_4**, a variável **Controle_4** tem o bit 0 setado e a variável **Processador_4** passa a conter o valor 1. Com isso a rotina **IntRx4** é ativada e de acordo com o conteúdo da variável **Processador_4** a rotina **ConsomeMensagem_4** é direcionada a pesquisar a área **Envia1_4**, que é onde a mensagem foi escrita. No fim da rotina **IntRx4** a variável **Controle_4** tem o bit 0 clareado;

. ou ainda, se o processador 2 escreve na área **Envia2_3**, a variável **Controle_3** tem o bit 0 setado e a variável **Processador_3** passa a conter o valor 2. Com isso a rotina **IntRx3** é ativada e de acordo com o conteúdo da variável **Processador_3** a rotina **ConsomeMensagem_3** é direcionada a pesquisar a área **Envia2_3**, que é onde a mensagem foi escrita. No fim da rotina **IntRx3** a variável **Controle_3** tem o bit 0 clareado;

Foram testados todos os algoritmos das **Primitivas** do Núcleo e das rotinas de tratamento de interrupção através desta simulação simplificada. Os resultados obtidos foram bastante positivos em relação à operação real.

CAPÍTULO V

CONCLUSÃO

As etapas de teste do Núcleo foram executadas podendo-se afirmar o correto funcionamento do mesmo.

Apesar das possíveis ocorrências de Espera Indefinida e Bloqueio Perpétuo, as técnicas e conceitos utilizados na elaboração deste trabalho foram bastante satisfatórias e viabilizam uma implementação.

Alguns testes a mais devem ser realizados para que se consiga uma avaliação mais ampla dos efeitos destas técnicas. Por exemplo, o tamanho das áreas compartilhadas e dos "buffers", assim como a quantidade de "buffers" devem ser alterados para diversos tipos de programa, para uma avaliação do desempenho da comunicação entre os elementos de processamento. Ainda, o número de processos permitidos, o número de rotinas permitidas e também o "time-slice" dos processos devem ser testados para a verificação do nível de paralelismo conseguido na máquina.

Alguns programas de teste de algoritmos paralelos devem ser submetidos à execução para se determinar corretamente o potencial desta máquina.

REFERÊNCIAS BIBLIOGRÁFICAS

DEFT (1984), **"DEFT Pascal User's Guide"**, DEFT Systems INC.

DEITEL, Harvey M. (1984), **"An Introduction to Operating Systems"**, Addison-Wesley Publishing Company, 1a. edição revisada.

HAYES, John P., Mudge, Trevor (1989), "Hypercube Supercomputres", **Proceedings of the IEEE**, vol. 77, no. 2.

HOLT, R. C., Graham, G. S., Lazowska, E. D. e Scott, M. A. (1978), **"Structured Concurrent Programming with Operating Systems Aplications"**, Addison-Wesley Publishing Company.

LISTER, A. M. (1984), **"Fundamentals of Operating Systems"**, Macmillan Press.

MARKENSON, Lilian (S.D.), **"Introdução à Estrutura de Dados"**, Instituto de Matemática, UFRJ, Rio de Janeiro.

MENDES, Sueli (1984), **"Programação Concorrente: Mecanismos de Comunicação e Sincronização de Processos"**, Quarta Escola de Computação, Instituto de Matemática e Estatística, USP, São Paulo.

MOTOROLA Incorporated (1985), **"8-Bit Microprocessors & Periferal Data"**, pp. 3-233 a 3-265 e pp. 3-397 a 3-409.

PETERSON, Janes L. e Silberschatz, A. (1985), **"Operating Systems Concepts"**, Addison-Wesley Publishing Company, 2a. edição.

SOUZA, Alberto F. (1988), **"O Hardware de uma Máquina Paralela Híbrida"**, Projeto de fim de curso da Escola de Engenharia, UFRJ, Rio de Janeiro.

APÊNDICE A

LISTAGEM DO PROGRAMA FONTE DO NÚCLEO

```
PROGRAM NUCLEOMPH;
(* NUC6.PAS - Versao Compacta para o COLOR64 *)

CONST
    NumMens = 10;
    NumProc = 6;
    NumBuf  = 8;
    TamFixo = 5;
    TamArea = 50;
    Pdores  = 4;              (* Numero de processadores *)
    NumRot  = 5;
    TimeSlice = 10;

                              (* Estados dos Processos *)
    Executando = 1;
    Pronto     = 2;
    BloqMens   = 3;
    BloqTempo  = 4;
    BloqBuffer = 5;
    BloqCompar = 6;
    BloqCriacao = 7;

                              (* Numero de Erros *)
    FaltaBuffer = 1;          (* Nao tem buffer disponivel *)
    FaltaArea   = 2;          (* Nao tem area disponivel *)
    ExcessoProc = 3;          (* Excedeu o limite de pro-
                                 cessos no processador *)
    ProcIne     = 4;          (* Processo nao existe na
                                 maquina *)

TYPE
    PtrInt = ^INTEGER;
    TipoArray = ARRAY [0..0] OF INTEGER;
    PtrArray = ^TipoArray;

                              (* Lista com ponteiro para o
                                 inicio e fim *)
    ListaDPont = RECORD
                 Inicio : INTEGER;
                 Fim    : INTEGER;
                 END;
    PtrLDPont = ^ListaDPont;

                              (* Lista com ponteiro somente
                                 para o inicio *)
    ListaPont = RECORD
                Inicio : INTEGER;
                END;
    PtrLPont  = ^ListaPont;

                              (* Descritor de Processo *)
    DescProc  = RECORD
                Proximo       : INTEGER;
                NomeProc      : INTEGER;
                Processador   : INTEGER;
                Pai           : INTEGER;
```

```
                Filhos          : INTEGER;
                LocalMem        : INTEGER;
                LocalDado       : INTEGER;
                LocalPilha      : INTEGER;
                Estado          : INTEGER;
                                (* # proc do qual quer receber
                                   mensagem *)
                ProcComunica    : INTEGER;
                EndMensagem     : PtrArray;
                (* Contexto        :         ; *)
                ListaMens       : ListaDPont;
                IntervEspera    : INTEGER;
                FatiaTempo      : INTEGER;
                END;


                                (* Celula da Tabela de Roti-
                                   nas *)
    CelTabela = RECORD
                Nome      : INTEGER;
                EndCarga  : INTEGER;
                TamCarga  : INTEGER;
                EndPilha  : INTEGER;
                TamPilha  : INTEGER;
                EndExec   : INTEGER;
                END;

                                (* Buffer de Mensagem *)
    Buffer    = RECORD
                Ponteiro  : INTEGER;
                Cadeia    : INTEGER;
                Mens      : ARRAY [0..4(*TamFixo-1*)] OF
                            INTEGER;
                END;

                                (* Banco da Memoria Comparti-
                                   lhada *)
    MemComp   = RECORD
                UltPosCheia    : INTEGER;
                UltPosLivre    : INTEGER;
                AreaMens       : ARRAY [0..49(*TamArea -
                                 1*)] OF INTEGER;
                MensConsumida  : INTEGER;
                END;
    PtrMemC   =  ^MemComp;


VAR
                                (* Lista de Descritores de
                                   Processo *)
    ListaDesc    : ARRAY [1..NumProc] OF DescProc;

                                (* Tabelas de Processos *)
                                (* para o processador 1 *)
    TabProc1     : ARRAY [1..Pdores,1..NumProc] OF INTEGER;
                                (* para o processador 2 *)
    TabProc2     : ARRAY [1..Pdores,1..NumProc] OF INTEGER;

                                (* Tabelas de Rotas *)
                                (* para o processador 1 *)
```

```
    TabRota1   : ARRAY [1..Pdores] OF INTEGER;
                                (* para o processador 2 *)
    TabRota2   : ARRAY [1..Pdores] OF INTEGER;

                                (* Tabela de Rotinas *)
    TabRotinas : ARRAY [1..NumRot] OF CelTabela;
                                (* Filas do Sistema *)
                                (* Processos Prontos *)
    ProcProntos : ListaDPont;
                                (* Espera por Intervalo *)
    BloqInterv  : ListaPont;
                                (* Espera por Mensagem *)
    BloqEMens   : ListaPont;
                                (* Bloqueados por Buffer *)
    BloqBuf     : ListaDPont;
                                (* Bloqueados por Area Compar-
                                   tilhada *)
    BloqComp    : ListaDPont;
                                (* Bloqueados por Criacao de
                                   Processos *)
    BloqCria    : ListaPont;

                                (* descritores nao utilizados *)
    DescVazios     : ListaDPont;

    ProcAtual    : INTEGER;  (* processo que esta executan-
                                do, e' quem pede servicos
                                ao nucleo *)
    DescAtual    : INTEGER;  (* descritor do processo que
                                esta' executando *)
    NumeroProc   : INTEGER;  (* identificador de proces-
                                sos *)
    Processador  : INTEGER;  (* numero do processador *)

    ParaConsumir : BOOLEAN;  (* indica que existe mensagem
                                na area compartilhada para
                                ser consumida *)

                                (* Area de Buffers de Mensa-
                                   gens *)
    AreaBuf        : ARRAY [1..NumBuf] OF Buffer;
                                (* Lista de Buffers Livres *)
    BufLivre       : ListaPont;
    QuantLivres : INTEGER;   (* quantidade de buffers li-
                                vres *)

(*** Variaveis para Simulacao ***)

                                (* Simulacao das memorias com-
                                   partilhadas *)
    Envia12 : MemComp;       (* Area onde processador 1
                                envia para o 2 *)
    Envia14 : MemComp;       (* Area onde processador 1
                                envia para o 4 *)
    Envia23 : MemComp;       (* Area onde processador 2
                                envia para o 3 *)

                                (* Simula a maquina (inter-
                                   rupcoes)  *)
    Controle1 : INTEGER;
```

```
    Controle2 : INTEGER;
    Controle3 : INTEGER;
    Controle4 : INTEGER;

    Processador1 : INTEGER;  (* simula o Flip-Flop dizendo
                                qual processador enviou a
                                mensagem *)
    Processador2 : INTEGER;
    Processador3 : INTEGER;
    Processador4 : INTEGER;

                             (* variaveis para teste da
                                logica *)
    message1 : ARRAY [0..30] OF INTEGER;
    message2 : ARRAY [0..30] OF INTEGER;
    TErro    : INTEGER;
    indice : INTEGER;
    tamanho : INTEGER;


(*              ROTINAS DE INICIALIZACAO                    *)
(**********************************************************)
procedure Inicializa;
var
   indice, auxiliar : INTEGER;

begin
                             (* Lista de descritores *)
   for indice := 1 to NumProc  do
   begin
      ListaDesc[indice].Proximo := indice + 1;
      ListaDesc[indice].Pai := 0;
      ListaDesc[indice].Filhos := 0;
      ListaDesc[indice].Estado := 0;
      ListaDesc[indice].ProcComunica := 0;
                             (* Fila de mensagem do Pro-
                                cesso *)
      ListaDesc[indice].ListaMens.inicio := 0;
      ListaDesc[indice].ListaMens.fim := 0;
      ListaDesc[indice].IntervEspera := 0;
      ListaDesc[indice].FatiaTempo := 0;
   end;
   ListaDesc[NumProc].Proximo := 0;
   DescVazios.inicio := 1;
   DescVazios.fim := NumProc;

   NumeroProc := 1;
   Processador := 1;

   DescAtual := 0;
   ProcAtual := 0;

   for indice := 1 to NumRot do
      TabRotinas[indice].Nome := 0;

   ProcProntos.inicio := 0;  (* Fila de Processos Prontos *)
   ProcProntos.fim := 0;

   BloqInterv.inicio := 0;   (* Fila de Processos Bloquea-
                                dos por Temporizacao *)
```

```
   BloqEMens.inicio := 0;      (* Fila de Processos Bloquea-
                                  dos a espera de mensagem *)
   BloqBuf.inicio := 0;        (* Fila de Processos Bloquea-
                                  dos a espera de buffer *)
   BloqBuf.fim := 0;

   BloqComp.inicio := 0;       (* Fila de Processos Bloquea-
                                  dos a espera de area com-
                                  partilhada *)
   BloqComp.fim := 0;

   BloqCria.inicio := 0;       (* Fila de Processos Bloquea-
                                  dos pela Criacao de Novos
                                  Processos *)


                               (* Encadeia os buffers, coloca
                                  na lista de buffers li-
                                  vres *)
   for indice := 1 to (NumBuf - 1) do
      AreaBuf[indice].Ponteiro := indice + 1;
   AreaBuf[indice].Ponteiro := 0;
   BufLivre.Inicio := 1;
   QuantLivres := NumBuf;

(***  Inicializacoes para o teste  ***)

                               (* Tabela de Processo / pro-
                                  cessador 1 *)
   for auxiliar := 1 to Pdores do
      for indice := 1 to NumProc do
      begin
         TabProc1[auxiliar,indice] := 0;
      end;
   TabRota1[1] := 0;
   TabRota1[2] := 2;
   TabRota1[3] := 2;
   TabRota1[4] := 4;

                               (* Tabela de Processo / pro-
                                  cessador 2 *)
   for auxiliar := 1 to Pdores do
      for indice := 1 to NumProc do
      begin
         TabProc2[auxiliar,indice] := 0;
      end;
   TabRota2[1] := 1;
   TabRota2[2] := 0;
   TabRota2[3] := 3;
   TabRota2[4] := 3;

                               (* Area de mensagem *)
   with Envia12 do begin       (* Area de Mensagem Vazia *)
      UltPosCheia := TamArea - 1;
      UltPosLivre := TamArea - 1;
      MensConsumida := 0;
   end;
```

```
   with Envia14 do begin
      UltPosCheia := TamArea - 1;
      UltPosLivre := TamArea - 1;
      MensConsumida := 0;
   end;

   with Envia23 do begin
      UltPosCheia := TamArea - 1;
      UltPosLivre := TamArea - 1;
      MensConsumida := 0;
   end;

                                (* Testando *)
   Controle1 := 0;
   Controle2 := 0;
   Controle3 := 0;
   Controle4 := 0;

   writeln ('Inicializou');

end;


(*  ROTINAS DE MANIPULACAO DE FILAS  *)
%C FILA/PAS:1


(*  ROTINAS GERAIS  *)
%C GERAL/PAS:1



(*  ROTINA DE ESCALONAMENTO DE PROCESSOS  *)
%C ESCALO/PAS:1


(*  ROTINAS DE TRATAMENTO DE INTERRUPCAO  *)
%C INT/PAS:1


(*  PRIMITIVAS DO NUCLEO  *)
%C PRIMI/PAS:1

(*********************************************************)
(*                  PROGRAMA PRINCIPAL                   *)
begin

   Inicializa;
   message1[0] := 3;
   message1[1] := 1;
   message1[2] := 1;
   CriaProc (3, 0, PTR(@message1), TErro);
   if TErro <> 0
      then writeln ('Criou todos os processos ');
   message1[0] := 1;          (* instancia *)
   message1[1] := 2;          (* rotina  *)
   message1[2] := 2;          (* processador *)
   message1[3] := 1;
   message1[4] := 3;
   message1[5] := 2;
```

```
 (*  CriaProc (2, ProcAtual, PTR(@message1), TErro);
   CriaProc (2, ProcAtual, PTR(@message1), Terro);*)
   Processador := 2;
   CriaProc (2, ProcAtual, PTR(@message1), TErro);
   TabProc2[2,1] := 8708;
   TabProc2[2,2] := 8965;
   if TErro <> 0
      then writeln ('Criou todos os processos ');
   Processador := 1;
   message1[0] := 30;
   for indice := 1 to message1[0] do
      message1[indice] := message1[0];

   Envia ( 1, 1, 3, PTR(@message1), TErro);
   Envia ( 2, 1, 2, PTR(@message1), TErro);
   Envia ( 2, 2, 4, PTR(@message1), TErro);
   Envia ( 2, 3, 5, PTR(@message1), TErro);
(*   Recebe (PTR(@message2), 1, 1, 1);*)
   Recebe ( PTR(@message2), 1, 1, 2);
   Espera (10);
   for indice := 1 to message2[0] do
      message2[indice] := 0;
   Recebe (PTR(@message2), 1, 1, 2);
   Termina;
   Termina;
   while DescAtual = 0 do
      Maquina;
   Envia ( 1, 1, 3, PTR(@message1), TErro);
   Recebe (PTR(@message2), 1, 1, 1);

end.
```

```
(*  FILA.PAS *)
(*              ROTINAS DE MANIPULACAO DE FILAS              *)
(***************************************************************)
procedure InsLDPont (ender : PtrLDPont; desc,
                     estado : INTEGER);
begin
   writeln ('Insere descritor LDPont');
   writeln (desc, estado);
   if ender^.inicio = 0
      then ender^.inicio := desc
      else ListaDesc[ender^.fim].Proximo := desc;
   ender^.fim := desc;
   writeln (ender^.inicio);
   ListaDesc[desc].Proximo := 0;
   ListaDesc[desc].Estado := estado;
end;


(***************************************************************)
function RetLDPont (ender : PtrLDPont) : INTEGER;
begin
   writeln ('Retira LDPont', ender^.inicio);
   if ender^.inicio <> 0
   then begin
      RetLDPont := ender^.inicio;
      ender^.inicio := ListaDesc[ender^.inicio].Proximo;
      if ender^.inicio = 0
         then ender^.fim := 0
   end
   else RetLDPont := 0;
   writeln (ender^.inicio);
end;


(***************************************************************)
procedure InsLPont (ender : PtrLPont; desc,
                    estado : INTEGER);
begin
   writeln ('Insere descritor LPont');
   writeln (desc, estado);
   if ender^.inicio = 0
      then ListaDesc[desc].Proximo := 0
      else ListaDesc[desc].Proximo := ender^.inicio;
   ender^.inicio := desc;
   writeln (ender^.inicio);
   ListaDesc[desc].Estado := estado;
end;



(***************************************************************)
procedure RetLPont ( ender : PtrLDPont; desc : INTEGER);
var
   anterior, auxiliar : INTEGER;
begin
   writeln ('Retira descritor LDPont');
   writeln (desc);
   if ender^.inicio <> desc
   then begin
```

```
      anterior := ender^.inicio;
      auxiliar := ListaDesc[anterior].Proximo;
      while auxiliar <> desc do
      begin
         anterior := auxiliar;
         auxiliar := ListaDesc[auxiliar].Proximo;
      end;
      ListaDesc[anterior].Proximo :=
         ListaDesc[auxiliar].Proximo;
   end
   else ender^.inicio := ListaDesc[ender^.inicio].Proximo;
   ListaDesc[desc].Proximo := 0;
   writeln (ender^.inicio);
end;


(*******************************************************)
procedure InsBloqInterv ( desc : INTEGER );
var
   primeiro, anterior, auxiliar : INTEGER;
begin
   writeln ('Insere BloqInterv');
   writeln (desc);
   if BloqInterv.inicio = 0
      then begin
         BloqInterv.inicio := desc;
         ListaDesc[desc].Proximo := 0;
      end
      else begin
         primeiro := BloqInterv.inicio;
         anterior := primeiro;
         auxiliar := primeiro;
         while auxiliar <> 0 (* fazer ate auxiliar = 0 *)
         do begin
            if ListaDesc[desc].IntervEspera <
                     ListaDesc[auxiliar].IntervEspera
               then begin
                  ListaDesc[desc].Proximo := auxiliar;
                  ListaDesc[auxiliar].IntervEspera :=
                     ListaDesc[auxiliar].IntervEspera -
                        ListaDesc[desc].IntervEspera;
                  if auxiliar = primeiro
                     then BloqInterv.inicio := desc
                     else ListaDesc[anterior].Proximo :=
                              desc;
                  auxiliar := 0;
               end
               else begin
                  ListaDesc[desc].IntervEspera :=
                     ListaDesc[desc].IntervEspera -
                        ListaDesc[auxiliar].IntervEspera;
                  if ListaDesc[auxiliar].Proximo <> 0
                     then begin
                        anterior := auxiliar;
                        auxiliar :=
                           ListaDesc[auxiliar].Proximo;
                     end
                     else begin
                        ListaDesc[auxiliar].Proximo := desc;
                        ListaDesc[desc].Proximo := 0;
```

```
                        if BloqInterv.inicio = 0
                           then BloqInterv.inicio := desc;
                        auxiliar := 0;
                     end;
               end;
         end;
      end;
   ListaDesc[desc].Estado := BloqTempo;
end;


(*********************************************************)
function RetBloqInterv : INTEGER;
begin
   writeln ('Retira BloqInterv', BloqInterv.inicio);
   RetBloqInterv := BloqInterv.inicio;
   BloqInterv.inicio :=
      ListaDesc[BloqInterv.inicio].Proximo;
end;



(*********************************************************)
procedure InsListaMens ( indicebuf, desc : INTEGER );
begin
   writeln ('Insere Mensagem');
   writeln (indicebuf, desc);
   AreaBuf[indicebuf].Ponteiro := 0;
   if ListaDesc[desc].ListaMens.fim = 0
      then ListaDesc[desc].ListaMens.inicio := indicebuf
      else AreaBuf[ListaDesc[desc].ListaMens.Fim].Ponteiro
           := indicebuf;
   ListaDesc[desc].ListaMens.fim := indicebuf;
end;


(*********************************************************)
function RetListaMens ( desc, origem : INTEGER ) : INTEGER;
var
   anterior, auxiliar : INTEGER;
begin
   writeln ('RetListaMens');
   writeln (desc, origem);
   if origem = 0
   then begin
      RetListaMens := ListaDesc[desc].ListaMens.inicio;
      ListaDesc[desc].ListaMens.inicio :=
         AreaBuf[ListaDesc[desc].ListaMens.inicio].Ponteiro;
      if ListaDesc[desc].ListaMens.inicio = 0
         then ListaDesc[desc].ListaMens.fim := 0;
   end
   else begin
      anterior := ListaDesc[desc].ListaMens.inicio;
      auxiliar := anterior;
      while ( AreaBuf[auxiliar].Mens[1] <> origem ) AND
            ( AreaBuf[auxiliar].Ponteiro <> 0 ) do
      begin
         anterior := auxiliar;
         auxiliar := AreaBuf[anterior].Ponteiro;
```

```
      end;
      if AreaBuf[auxiliar].Mens[1] = origem
      then begin
         RetListaMens := auxiliar;
         if ListaDesc[desc].ListaMens.inicio = auxiliar
            then ListaDesc[desc].ListaMens.inicio :=
                 AreaBuf[auxiliar].Ponteiro
            else AreaBuf[anterior].Ponteiro :=
                 AreaBuf[auxiliar].Ponteiro;
         if ListaDesc[desc].ListaMens.fim = auxiliar
            then ListaDesc[desc].ListaMens.fim := anterior;
         if ListaDesc[desc].ListaMens.inicio = 0
            then ListaDesc[desc].ListaMens.fim := 0;
      end
      else RetListaMens := 0;
   end;
end;


(*********************************************************)
procedure InsBloqBuf ( desc : INTEGER );
begin
   writeln ('Insere BloqBuf');
   writeln (desc);
   if BloqBuf.inicio = 0
      then BloqBuf.inicio := desc
      else ListaDesc[BloqBuf.fim].Proximo := desc;
   BloqBuf.fim := desc;
   ListaDesc[desc].Proximo := 0;
   ListaDesc[desc].Estado := BloqBuffer;
   writeln (BloqBuf.inicio);
end;



(*********************************************************)
procedure RetBloqBuf (desc : INTEGER);
var
   auxiliar, anterior : INTEGER;
begin
   writeln ('Retira BloqBuf');
   writeln ('desc');
   if desc = BloqBuf.inicio
   then begin
      BloqBuf.inicio := ListaDesc[desc].Proximo;
      if BloqBuf.inicio = 0
         then BloqBuf.fim := 0;
   end
   else begin
      anterior := BloqBuf.inicio;
      auxiliar := ListaDesc[anterior].Proximo;
      while auxiliar <> desc do
      begin
         anterior := auxiliar;
         auxiliar := ListaDesc[auxiliar].Proximo;
      end;
      ListaDesc[anterior].Proximo :=
         ListaDesc[desc].Proximo;
      if desc = BloqBuf.fim
         then BloqBuf.fim := anterior;
```

```
   end;
   ListaDesc[desc].Proximo := 0;
   writeln (BloqBuf.inicio, desc);
end;


(*******************************************************)
procedure InsBloqComp ( desc : INTEGER );
begin
   writeln ('Insere BloqComp');
   writeln (desc);
   if BloqComp.inicio = 0
      then BloqComp.inicio := desc
      else ListaDesc[BloqComp.fim].Proximo := desc;
   BloqComp.fim := desc;
   ListaDesc[desc].Proximo := 0;
   ListaDesc[desc].Estado := BloqCompar;
   writeln (BloqComp.inicio);
end;


(*******************************************************)
procedure RetBloqComp (desc : INTEGER);
var
   auxiliar, anterior : INTEGER;
begin
   writeln ('Retira BlqComp');
   writeln (desc);
   if desc = BloqComp.inicio
   then begin
      BloqComp.inicio := ListaDesc[desc].Proximo;
      if BloqComp.inicio = 0
         then BloqComp.fim := 0;
   end
   else begin
      anterior := BloqComp.inicio;
      auxiliar := ListaDesc[anterior].Proximo;
      while auxiliar <> desc do
      begin
         anterior := auxiliar;
         auxiliar := ListaDesc[auxiliar].Proximo;
      end;
      ListaDesc[anterior].Proximo :=
         ListaDesc[desc].Proximo;
      if desc = BloqComp.fim
         then BloqComp.fim := anterior;
   end;
   ListaDesc[desc].Proximo := 0;
   writeln (BloqComp.inicio, desc);
end;
```

```
(*   GERAL.PAS  *)
(*                        ROTINAS GERAIS                          *)
(****************************************************************)
procedure PesqTabProc1 (pdor, proc : INTEGER; var lugar,
                        erro : INTEGER);
var
   auxiliar : INTEGER;
begin
   writeln ('Pesquisa TabProc1');
   erro := 1;
   for auxiliar := 1 to NumProc do
   begin
      if TabProc1[pdor,auxiliar] = proc
         then begin
            erro := 0;
            lugar := TabRota1[pdor];
            writeln (proc, lugar);
            auxiliar := NumProc;
         end;
   end;
end;


(****************************************************************)
procedure PesqTabProc2 (pdor, proc : INTEGER; var lugar,
                        erro : INTEGER);
var
   auxiliar : INTEGER;
begin
   writeln ('Pesquisa TabProc2');
   erro := 1;
   for auxiliar := 1 to NumProc do
   begin
      if TabProc2[pdor,auxiliar] = proc
         then begin
            erro := 0;
            lugar := TabRota2[pdor];
            writeln (proc, lugar);
            auxiliar := NumProc;
         end;
   end;
end;



(****************************************************************)
function DescobreDesc (proc : INTEGER) : INTEGER;
var
   indice : INTEGER;
begin
   writeln ('DescobreDesc', proc);
   indice := 0;
   repeat indice := indice + 1
      until ListaDesc[indice].NomeProc = proc;
   DescobreDesc := indice;
end;


(****************************************************************)
```

```
procedure CopiaMens (destino : PtrArray; inidest : INTEGER;
                     origem : PtrArray; iniorig,
                     tamanho : INTEGER);
var
   indice : INTEGER;
begin
   writeln ('Copia mensagem');
   for indice := 0 to (tamanho - 1)
      do destino^[indice+inidest] :=
            origem^[indice+iniorig];
end;



(*      ROTINAS DE MANIPULACAO DA AREA DE MENSAGENS      *)
(*******************************************************)
function AlocaBuf : INTEGER;
begin
   writeln ('AlocaBuf', BufLivre.Inicio);
   AlocaBuf := BufLivre.Inicio;
   BufLivre.Inicio := AreaBuf[BufLivre.Inicio].Ponteiro;
   QuantLivres := QuantLivres - 1;
end;


(*******************************************************)
procedure DevolveBuf (indbuf : INTEGER);
begin
   writeln ('Devolve Buffer');
   writeln (indbuf);
   AreaBuf[indBuf].Ponteiro := BufLivre.Inicio;
   BufLivre.Inicio := indBuf;
   QuantLivres := QuantLivres + 1;
end;



(*******************************************************)
procedure GuardaMens (dest, orig : INTEGER; mens : PtrArray;
                      comp : boolean; posicao : INTEGER;
                      var aponta, erro : INTEGER);
var
   numbuf, indicebuf, elemento, auxiliar, total : INTEGER;
   copia, sobra, tamanho, quant : INTEGER;
   primeiro : boolean;

begin
   writeln ('Guarda Mensagem');
   writeln ( dest, orig, ORD(mens), comp, posicao);
   erro := 0;
   primeiro := true;
   if comp
      then tamanho := mens^[posicao] + 1
      else tamanho := mens^[0] + 1;
   total := tamanho + 2;      (* calcula quantidade de
                                 buffers necessarios para
                                 armazenar mensagem *)
   numbuf := total DIV TamFixo;
   if (total MOD TamFixo) > 0
```

```
   then numbuf := numbuf + 1;
                        (* se existe buffers sufi-
                           cientes *)
if (QuantLivres >= numbuf)
   then
      while numbuf > 0 do
         begin
            indicebuf := AlocaBuf;
            if primeiro   (* usa o primeiro buffer *)
               then begin
                        (* guarda o indice do primeiro
                           buffer *)
                  aponta := indicebuf;
                        (* coloca cabecalho *)
                  AreaBuf[indicebuf].Ponteiro := 0;
                  AreaBuf[indicebuf].Mens[0] := dest;
                  AreaBuf[indicebuf].Mens[1] := orig;
                        (* determina o tamanho a co-
                           piar *)
                  if total > TamFixo
                     then begin
                        copia := TamFixo - 2;
                        sobra := tamanho - copia;
                     end
                     else begin
                        copia := tamanho;
                        AreaBuf[indicebuf].Cadeia := 0;
                     end;
                        (* comeca a copiar mensagem *)
                  if comp
                  then begin
                     if (posicao + copia) > TamArea
                     then begin
                        quant := TamArea - posicao;
                        CopiaMens
                        (PTR(@AreaBuf[indicebuf].Mens),
                         2, mens, posicao, quant);
                        posicao := (posicao + quant) MOD
                                   TamArea;
                        CopiaMens
                        (PTR(@AreaBuf[indicebuf].Mens),
                         (2 + quant), mens, posicao,
                         (copia - quant));
                     end
                     else begin
                        CopiaMens
                        (PTR(@AreaBuf[indicebuf].Mens),
                         2, mens, posicao, copia);
                        posicao := (posicao + copia) MOD
                                   TamArea;
                     end
                  end
                  else begin
                     CopiaMens
                     (PTR(@AreaBuf[indicebuf].Mens), 2,
                      mens, 0, copia);
                        (* incrementa no array *)
                     elemento := copia;
                  end;
                  primeiro := false;
```

```
                    end
                    else begin (* se existirem buffers res-
                                  tantes, encadeia o buffer
                                  anterior *)
                       AreaBuf[auxiliar].Ponteiro :=
                          indicebuf;
                       AreaBuf[auxiliar].Cadeia := indicebuf;
                              (* determina o tamanho a
                                 copiar *)
                       if sobra > TamFixo
                          then begin
                             copia := TamFixo;
                             sobra := sobra - copia;
                          end
                          else begin
                             copia := sobra;
                             AreaBuf[indicebuf].Ponteiro := 0;
                             AreaBuf[indicebuf].Cadeia := 0;
                          end;
                       if comp
                       then begin
                          if (posicao + copia) > TamArea
                          then begin
                             quant := TamArea - posicao;
                             CopiaMens
                             (PTR(@AreaBuf[indicebuf].Mens),
                              0, mens, posicao, quant);
                             posicao := (posicao + quant) MOD
                                        TamArea;
                             CopiaMens
                             (PTR(@AreaBuf[indicebuf].Mens),
                              (0 + quant), mens, posicao,
                              (copia - quant));
                             posicao := posicao + copia -
                                        quant;
                          end
                          else begin
                             CopiaMens
                             (PTR(@AreaBuf[indicebuf].Mens),
                              0, mens, posicao, copia);
                             posicao := (posicao + copia) MOD
                                        TamArea;
                          end
                       end
                              (* copia parte da mensagem *)
                       else begin
                          CopiaMens
                          (PTR(@AreaBuf[indicebuf].Mens), 0,
                           mens, elemento, copia);
                              (* incrementa no array *)
                          elemento := elemento + copia;
                       end;
                    end;
                 numbuf := numbuf - 1;
                 auxiliar := indicebuf;
              end
                          (* nao tem buffers necessarios *)
        else erro := FaltaBuffer;
end;
```

```
(*          ROTINAS PARA MANIPULACAO DA AREA DE
                    MEMORIA COMPARTILHADA                    *)
(*************************************************************)
procedure InsereMens (areamem : PtrMemC; dest,
                      orig : INTEGER; ender : PtrArray;
                      posicao : INTEGER; comp : boolean;
                      tammsg : INTEGER; var erro : INTEGER);
var
   livres, tamanho, quant, controle : INTEGER;
   aux : PtrArray;
begin
   writeln ('Insere Mensagem');
   writeln (ORD(areamem), dest, orig);
   writeln (ORD(ender), posicao, comp, tammsg);
                              (* Area de Mensagem Vazia *)
   if areamem^.UltPosCheia = areamem^.UltPosLivre
      then livres := TamArea
      else
         if areamem^.UltPosCheia > areamem^.UltPosLivre
            then livres := TamArea - (areamem^.UltPosCheia
                           - areamem^.UltPosLivre)
            else livres := areamem^.UltPosLivre -
                           areamem^.UltPosCheia - 1;
   if livres >= tammsg
   then begin
      if dest <> 0
      then begin
         areamem^.UltPosCheia := (areamem^.UltPosCheia + 1)
                                 MOD TamArea;
         areamem^.AreaMens[areamem^.UltPosCheia] := dest;
         areamem^.UltPosCheia := (areamem^.UltPosCheia + 1)
                                 MOD TamArea;
         areamem^.AreaMens[areamem^.UltPosCheia] := orig;
         tamanho := tammsg + 1;
      end
      else tamanho := tammsg;
      indice := areamem^.UltPosCheia;
      aux := PTR(@areamem^.AreaMens);
      for controle := 0 to (tamanho-1) do
      begin
         indice := (indice + 1) MOD TamArea;
         aux^[indice] := ender^[posicao];
         if comp
            then posicao := (posicao + 1) MOD TamArea
            else posicao := posicao + 1;
      end;
      areamem^.UltPosCheia := indice;
      Controle2 := Controle2 OR 1;
      Processador2 := 1;
   end
   else erro := FaltaArea; (* falta area para a mensagem *)
end;


(*************************************************************)
procedure VBloqComp (lado : INTEGER);
var
   desc, erro : INTEGER;
begin
```

```
   writeln ('Ver BloqComp');
   writeln (lado);
   desc := BloqComp.inicio;
   if lado = 1
   then begin
      erro := 0;
      while (BloqComp.inicio <> 0) AND (erro = 0) do
      begin
         InsereMens (PTR(@Envia12),
                     ListaDesc[desc].ProcComunica,
                     ListaDesc[desc].NomeProc,
                     ListaDesc[desc].EndMensagem, 0, false,
                     ListaDesc[desc].EndMensagem^[0], erro);
         if erro = 0
         then begin
            RetBloqComp (desc);
            InsLDPont (PTR(@ProcProntos), desc, Pronto);
         end
      end
   end
   else begin                    (* lado = 3 *)
   end
end;


(*******************************************************)
procedure VBloqBuf;
var
   tamanho, erro, aponta, desc : INTEGER;
begin
   writeln ('Ver BloqBuf');
   if ParaConsumir
   then begin
      with Envia12 do
      begin
         erro := 1;
         while (UltPosLivre <> UltPosCheia) AND (erro <> 0)
         do begin
            tamanho := AreaMens[(UltPosLivre + 3) MOD
                       TamArea] + 3;
            GuardaMens (AreaMens[(UltPosLivre+1) MOD
                        TamArea], AreaMens[(UltPosLivre+2)
                        MOD TamArea], PTR(@AreaMens), true,
                        (UltPosLivre + 3) MOD TamArea,
                        aponta, erro);
            if erro = 0
            then begin
               desc := DescobreDesc
                       (AreaMens[(UltPosLivre+1) MOD
                        TamArea]);
               InsListaMens (aponta, desc);
               UltPosLivre := (UltPosLivre + tamanho) MOD
                              TamArea;
               if UltPosLivre = UltPosCheia
                  then ParaConsumir := false;
               if BloqComp.inicio <> 0
                  then VBloqComp (1);
            end
         end
      end
```

```
    end;
    if (BloqBuf.inicio <> 0) AND (QuantLivres <> 0)
    then begin
       erro := 1;
       indice := BloqBuf.inicio;
       while (indice <> 0) AND (erro <> 0) do
       begin
          GuardaMens (ListaDesc[indice].ProcComunica,
                      ListaDesc[indice].NomeProc,
                      ListaDesc[indice].EndMensagem, false,
                      0, aponta, erro);
          if erro = 0
          then begin
             desc := DescobreDesc
                     (ListaDesc[indice].ProcComunica);
             InsListaMens (aponta, desc);
             RetBloqBuf (indice);
             InsLDPont (PTR(@ProcProntos), indice, Pronto);
          end
          else indice := ListaDesc[indice].Proximo;
       end
    end
end;
```

```
(*   ESCALO.PAS  *)
(*         ROTINA DE ESCALONAMENTO DE PROCESSOS          *)
(*********************************************************)
procedure Escalonador;
begin
   write ('Escalnador');
   (* salva contexto do processo atual *)
   if ProcProntos.inicio = 0
      then begin
         (* deve ficar um processo idle rodando *)
         DescAtual := 0;
         ProcAtual := 0;
                              (* para testar o relogio *)
         Controle1 := Controle1 OR 2;
      end
      else begin
         ListaDesc[ProcProntos.inicio].Estado :=
         Executando;
         DescAtual := ProcProntos.inicio;
         ProcAtual := ListaDesc[DescAtual].NomeProc;
         if ListaDesc[DescAtual].FatiaTempo = 0
            then ListaDesc[DescAtual].FatiaTempo := 10;
            (* restaura o contexto do novo processo *)
      end;
   writeln (DescAtual, ProcAtual);
end;
```

```
(*   INT.PAS   *)
(*          ROTINAS DE TRATAMENTO DE INTERRUPCAO          *)
(*********************************************************)
procedure TrataRelogio1;
var
   desc : INTEGER;
begin
   write ('TrataRelogio   ');
   if ProcAtual <> 0
   then begin
      writeln ('time-slice');
      ListaDesc[DescAtual].FatiaTempo :=
         ListaDesc[DescAtual].FatiaTempo - 1;
      if ListaDesc[DescAtual].FatiaTempo = 0
      then begin
         desc := RetLDPont (PTR(@ProcProntos));
         InsLDPont (PTR(@ProcProntos), DescAtual, Pronto);
         Escalonador;
      end;
   end;
   if BloqInterv.inicio <> 0
   then begin
      writeln (' Intervalo ');
      ListaDesc[BloqInterv.inicio].IntervEspera :=
         ListaDesc[BloqInterv.inicio].IntervEspera - 1;
      while (ListaDesc[BloqInterv.inicio].IntervEspera = 0)
            AND (BloqInterv.inicio <> 0) do
      begin
         desc := RetBloqInterv;
         InsLDPont (PTR(@ProcProntos), desc, Pronto);
      end;
      if DescAtual = 0
         then Escalonador ;
   end;
   Controle1 := Controle1 AND 253;
end;


(*********************************************************)
procedure Roteia2 (pdordest, destino, origem : INTEGER;
                   ender : PtrArray; posicao,
                   tamanho : INTEGER; var erro : INTEGER);
begin
   writeln ('Roteia', pdordest, destino, origem);
   writeln (ORD(ender), posicao, tamanho);
   case TabRota2[pdordest] of
(*       1 : InsereMens (PTR(@Envia21), destino, origem,
                ender, posicao, true, tamanho, erro); *)
      3 : InsereMens (PTR(@Envia23), destino, origem, ender,
                      posicao, true, tamanho, erro)
   end;
end;


(*********************************************************)
procedure IntRx1;
begin
end;
```

```
(*********************************************************)
procedure AvisaPai (pai : INTEGER);
var
   desc : INTEGER;

begin
   writeln ('AvisaPai', pai);
   desc := DescobreDesc (pai);
   ListaDesc[desc].Filhos := ListaDesc[desc]. Filhos - 1;
   if ListaDesc[desc].Filhos = 0
   then begin
      RetLPont (PTR(@BloqCria), desc);
      InsLDPont (PTR(@ProcProntos), desc, Pronto);
   end;
end;



(*********************************************************)
procedure ConsomeMensagem2 (areamem : PtrMemC);
var
   aponta, erro, destino, tamanho, proc, livres : INTEGER;
   desc, par1, par2, par3, par4, par5 : INTEGER;
   msg : ARRAY [0..4] of INTEGER;
   roteia : boolean;

begin
   writeln ('ConsomeMensagem2', ORD(areamem));
   Processador := 2;
   erro := 0;
   while (areamem^.UltPosLivre <> areamem^.UltPosCheia) AND
         (erro = 0) do
   begin
      destino := areamem^.AreaMens[(areamem^.UltPosLivre +
                 1) MOD TamArea];
      par1 := areamem^.AreaMens[(areamem^.UltPosLivre + 2)
              MOD TamArea];
      par2 := areamem^.AreaMens[(areamem^.UltPosLivre + 3)
              MOD TamArea];
      if destino = 0
      then begin
         par3 := areamem^.AreaMens[(areamem^.UltPosLivre +
                 4) MOD TamArea];
         par4 := areamem^.AreaMens[(areamem^.UltPosLivre +
                 5) MOD TamArea];
         par5 := areamem^.AreaMens[(areamem^.UltPosLivre +
                 6) MOD TamArea];
         roteia := false;
         case par1 of
            10 : begin       (* carregamento de rotina*)
                    writeln ('carrega rotina');
                    if par2 = Processador
                    then begin
                       indice := 0;
                       repeat indice := indice + 1;
                          until TabRotinas[indice].Nome = 0;
                       TabRotinas[indice].Nome := par3;
                       TabRotinas[indice].EndCarga := par4;
```

```
                    TabRotinas[indice].TamCarga := par5;
                    TabRotinas[indice].EndPilha :=
                       areamem^.AreaMens[
                       (areamem^.UltPosLivre + 7) MOD
                       TamArea];
                    TabRotinas[indice].TamPilha :=
                       areamem^.AreaMens[
                       (areamem^.UltPosLivre + 8) MOD
                       TamArea];
                    TabRotinas[indice].EndExec :=
                       areamem^.AreaMens[
                       (areamem^.UltPosLivre + 9) MOD
                       TamArea];
                    (* carrega na memoria *)
                 end
                 else begin
                    roteia := true;
                    proc := par2;
                    tamanho := 9;
                 end;
              end;
        11 : begin       (* criacao de processos*)
                 writeln ('cria processo');
                 if par2 = Processador
                 then begin
                    msg[0] := par4;
                    msg[1] := par3;
                    msg[2] := Processador;
(*                    CriaProc (par4, par5, PTR(@msg),
                              erro); *)
                 end
                 else begin
                          (* roteia para o processador
                             destino *)
                    roteia := true;
                    proc := par2;
                 end;
                 tamanho := 6;
              end;
        12 : begin       (* inclusao de processo *)
                 writeln ('inclui processo');
                 if par2 <> Processador
                 then begin
                    indice := 0;
                    repeat indice := indice + 1
                       until TabProc2[par2,indice] = 0;
                    TabProc2[par2,indice] := par3;
                             (* roteia para o proximo
                                processador *)
                    roteia := true;
                    proc := (Processador + 1) MOD Pdores;
                    tamanho := 4;
                 end;
              end;
        13 : begin       (* termino de processo *)
                 writeln ('termina processo');
                 if Processador <> par2
                 then begin
                    indice := 0;
                    repeat indice := indice + 1
```

```
                        until TabProc2[par2,indice] =
                              par3;
                     TabProc2[par2,indice] := 0;
                     if (par4 <> 0) AND (((par4 LSR 13) +
                         1) = Processador)
                        then AvisaPai (par4);
                  end;
                  roteia := true;
                  proc := (Processador + 1) MOD Pdores;
                  tamanho := 5;
               end
      end;
      if roteia
         then Roteia2 (proc, 0, 0,
              PTR(@areamem^.AreaMens), (areamem^.
              UltPosLivre + 1) MOD TamArea, tamanho,
              erro);
      areamem^.UltPosLivre := (areamem^.UltPosLivre +
                               tamanho) MOD TamArea;
      (* avisa consumo de mensagem *)
   end
   else begin
                           (* se e' para um processo deste
                              processador *)
      if ((destino LSR 13) + 1) = Processador
      then begin
         erro := 0;
         desc := DescobreDesc (destino);
                           (* verifica se o processo es-
                              tava bloqueado a espera de
                              mensagem do processo ori-
                              gem *)
         if ListaDesc[desc].Estado = BloqMens
         then begin
            if ListaDesc[desc].ProcComunica = par1
            then begin
               CopiaMens (ListaDesc[desc].EndMensagem, 0,
                          PTR(@areamem^.AreaMens),
                          (areamem^.UltPosLivre +3) MOD
                          TamArea, (par2 + 1));
               RetLPont (PTR(@BloqEMens), desc);
               InsLDPont (PTR(@ProcProntos), desc,
                          Pronto);
            end;
         end
         else begin
            GuardaMens (destino, par1, PTR(@areamem^.
                        AreaMens), true, (areamem^.
                        UltPosLivre + 3) MOD TamArea,
                        aponta, erro);
            if erro = 0    (* se nao houve erro na Guar-
                              daMens encadeia mensagem
                              no descritor *)
               then  InsListaMens (aponta, desc)
                           (* espera ter buffer disponivel
                              para guardar mensagem *)
               else   ParaConsumir := true;
         end;
         (* libera a area compartilhada *)
         if erro = 0       (* se a mensagem foi absorvi-
```

```
                                da libera a area comparti-
                                lhada *)
                then areamem^.UltPosLivre := (areamem^.
                     UltPosLivre + par2 + 3) MOD TamArea;
          end
          else begin            (* e' para outro processador *)
             Roteia2 ( ((destino LSR 13) + 1), 0, 0,
                      PTR(@areamem^.AreaMens), areamem^.
                      UltPosCheia, tamanho, erro);
          end;
       end;
   end;
                                (* clareia a interrupcao *)
   Controle2 := Controle2 AND 254;
   Processador := 1;
end;


(*********************************************************)
procedure IntRx2;
begin
   writeln ('IntRx2');
   case Processador2 of
      1 : begin
             if Envia12.MensConsumida <> 0
             then begin
                if BloqComp.inicio <> 0
                   then VBloqComp (Processador);
             end
             else ConsomeMensagem2 (PTR(@Envia12));
          end;
      3 : begin                 (* area Envia31 *)
          end
   end;
end;


(*             ROTINAS DE SIMULACAO DA MAQUINA             *)
(*********************************************************)
procedure Maquina;
begin
   writeln ('Maquina');
                                (* conta tempo do processador
                                   1 *)
   Controle1 := Controle1 OR 2;
                                (* verifica se tem algum pro-
                                   cesso que pode ser desblo-
                                   queado por tempo *)
   if (Controle1 AND 2) = 2
      then TrataRelogio1;

                                (* verifica se ha mensagem
                                   para ser consumida *)
   if (Controle1 AND 1) = 1
      then IntRx1;

   if (Controle2 AND 1) = 1
      then IntRx2;

(*   if (Controle3 AND 1) = 1
```

```
      then IntRx3;

   if (Controle4 AND 1) = 1
      then IntRx4;    *)
end;
```

```
(*  PRIMI.PAS  *)
(*                    PRIMITIVAS DO NUCLEO                    *)
(************************************************************)
procedure Envia (pdor, rotina, instancia : INTEGER;
                 mens : PtrArray; var erro : INTEGER);
var
   dest, desc, lixo, lugar, aponta : INTEGER;
begin
   writeln ('Envia', pdor, rotina);
   writeln (instancia, ORD(mens));
   (* bloqueia as interrupcoes *)
   (* congela o relogio *)
   erro := 0;
   dest := ((pdor-1) LSL 13) + (rotina LSL 8) + instancia;
   PesqTabProc1 (pdor, dest, lugar, erro);
   if erro = 0
   then begin
      if pdor = Processador  (* o processo esta neste pro-
                                cessador *)
      then begin
                            (* descobre o descritor do des-
                               tino se o processo esta blo-
                               queado a espera de mensagem
                               e o processo destino quer
                               receber do processo que esta
                               enviando *)
         desc := DescobreDesc (dest);
         if ListaDesc[desc].Estado = BloqMens
         then begin
            if ListaDesc[desc].ProcComunica = ProcAtual
            then begin        (* entao libera o processo
                                 destino *)
               CopiaMens (ListaDesc[desc].EndMensagem, 0,
                          mens, 0, (mens^[0] + 1));
               RetLPont (PTR(@BloqEMens), desc);
               InsLDPont (PTR(@ProcProntos), desc, Pronto);
            end;
         end
         else begin
            GuardaMens (dest, ProcAtual, mens, false, 0,
                        aponta, erro);
            if erro = 0       (* se conseguiu guardar a men-
                                 sagem, encadeia a mensa-
                                 gem *)
               then InsListaMens (aponta, desc)
               else begin     (* erro = FaltaBuffer *)
                  lixo := RetLDPont (PTR(@ProcProntos));
                                 (* bloqueia o processo ate que
                                    possa enviar a mensagem *)
                  InsBloqBuf (DescAtual);
                  ListaDesc[DescAtual].ProcComunica :=
                     dest;
                  ListaDesc[DescAtual].EndMensagem := mens;
                  Escalonador;
               end
         end
      end
      else begin              (* se o processo esta em outro
                                 processador *)
```

```
         if lugar = 2
            then InsereMens (PTR(@Envia12), dest,
                             ProcAtual, mens, 0, false,
                             mens^[0], erro)
            else InsereMens (PTR(@Envia14), dest,
                             ProcAtual, mens, 0, false,
                             mens^[0], erro);
         if erro = FaltaArea (* nao ha area disponivel *)
            then begin
               lixo := RetLDPont (PTR(@ProcProntos));
                             (* bloqueia o processo ate que
                                possa enviar a mensagem *)
               InsBloqComp (DescAtual);
               ListaDesc[DescAtual].ProcComunica := dest;
               ListaDesc[DescAtual].EndMensagem := mens;
               Escalonador;
            end;
      end;
   end
   else erro := ProcIne;
   Maquina;
   (* descongela o relogio *)
   (* desbloqueia as interrupcoes *)
end;


(*******************************************************)
procedure Recebe (ender : PtrArray; pdor, rotina,
                  instancia : INTEGER);
var
   desc, buffer, anterior, copia, sobra : INTEGER;
   elemento, proc : INTEGER;

begin
   writeln ('Recebe', ORD(ender));
   writeln (pdor, rotina, instancia);
   (* bloqueia as interrupcoes *)
   (* congela o relogio *)
   proc := ((pdor-1) LSL 13) + (rotina LSL 8) + instancia;
                             (* se a fila de mensagens esta
                                vazia *)
   if ListaDesc[DescAtual].ListaMens.Inicio = 0
      then begin
         desc := RetLDPont (PTR(@ProcProntos));
                             (* bloqueia o processo *)
         InsLPont (PTR(@BloqEMens), DescAtual, BloqMens);
         ListaDesc[DescAtual].ProcComunica := proc;
         ListaDesc[DescAtual].EndMensagem := ender;
         Escalonador;
      end
      else begin
         buffer := RetListaMens (DescAtual,proc);
         if buffer <> 0
         then begin
            tamanho := AreaBuf[buffer].Mens[2];
            if AreaBuf[buffer].Cadeia <> 0
            then begin
               copia := TamFixo - 2;
               CopiaMens (ender, 0, PTR(@AreaBuf[buffer].
                          Mens), 2, copia);
```

```
                        (* incrementa no array *)
                elemento := copia;
                sobra := tamanho + 1 - copia;
                while AreaBuf[buffer].Cadeia <> 0 do
                begin
                   if sobra > TamFixo
                   then begin
                      copia := TamFixo;
                      sobra := sobra - copia;
                   end
                   else copia := sobra;
                   anterior := buffer;
                   buffer := AreaBuf[buffer].Cadeia;
                   DevolveBuf (anterior);
                   CopiaMens (ender, elemento, PTR(@AreaBuf
                              [buffer].Mens), 0, copia);
                              (* incrementa no array *)
                   elemento := elemento + copia;
                end;
                DevolveBuf(buffer);
             end
             else begin
                CopiaMens (ender, 0, PTR(@AreaBuf[buffer].
                           Mens), 2, tamanho);
                DevolveBuf(buffer);
             end;
                              (* veriifica os bloqueios *)
             if ParaConsumir OR (BloqBuf.inicio <> 0)
                then VBloqBuf;
          end
          else begin          (* a fila nao tem a mensagem
                                 desejada *)
             desc := RetLDPont (PTR(@ProcProntos));
             InsLPont (PTR(@BloqEMens), DescAtual,
                       BloqMens);
             ListaDesc[DescAtual].ProcComunica := proc;
             ListaDesc[DescAtual].EndMensagem := ender;
             Escalonador;
          end;
       end;
    Maquina;
    (* descongela o relogio *)
    (* desbloqueia as interrupcoes *)
end;


(*******************************************************)
procedure Espera (quantidade : INTEGER);
var
   desc : INTEGER;
begin
   writeln ('Espera', quantidade);
   (* bloqueia as interrupcoes *)
   (* congela o relogio *)
   ListaDesc[DescAtual].IntervEspera := quantidade;
   desc := RetLDPont (PTR(@ProcProntos));
   InsBloqInterv (DescAtual);
   (*ListaDesc[DescAtual].Estado := BloqTempo;*)
   Escalonador;
   Maquina;
```

```
   (* descongela o relogio *)
   (* desbloqueia as interrupcoes *)
end;


(*********************************************************)
procedure CriaProc (quant, pai : INTEGER; ListaRot :
                    PtrArray; var erro : INTEGER);
var
   desc, indice, ponteiro, instancias, rotina : INTEGER;
   numpdor, filhos : INTEGER;
   msg : ARRAY [0..5] OF INTEGER;
begin
   writeln ('CriaProc', quant, pai);
   filhos := 0;
   erro := 0;
   ponteiro := 0;
   while quant <> 0 do
   begin
      instancias := ListaRot^[ponteiro];
      ponteiro := ponteiro + 1;
      rotina := ListaRot^[ponteiro];
      ponteiro := ponteiro + 1;
      numpdor := ListaRot^[ponteiro];
      ponteiro := ponteiro + 1;
      if numpdor <> Processador
      then begin
         msg[0] := 0;
         msg[1] := 11;
         msg[2] := numpdor;
         msg[3] := rotina;
         msg[4] := instancias;
         msg[5] := ProcAtual;
         writeln (msg[2], msg[3], msg[4], msg[5]);
         if TabRota1[numpdor] = 2
            then InsereMens(PTR(@Envia12), 0, 0, PTR(@msg),
                            0, false, 6, erro)
            else InsereMens(PTR(@Envia14), 0, 0, PTR(@msg),
                            0, false, 6, erro);
         filhos := filhos + 1;
         quant := quant - instancias;
         instancias := 0;
      end
      else while (instancias <> 0) AND (erro = 0) do
      begin
         desc := RetLDPont (PTR(@DescVazios));
         instancias := instancias - 1;
         quant := quant - 1;
         if desc <> 0
         then begin
            filhos := filhos + 1;
            ListaDesc[desc].NomeProc := ((Processador - 1)
               LSL 13) + (rotina LSL 8) + NumeroProc;
            NumeroProc := NumeroProc + 1;
            ListaDesc[desc].Processador := Processador;
            ListaDesc[desc].Pai := pai;
            ListaDesc[desc].Estado := Pronto;
            ListaDesc[desc].ProcComunica := 0;
            ListaDesc[desc].ListaMens.inicio := 0;
            ListaDesc[desc].ListaMens.fim := 0;
```

```
            ListaDesc[desc].IntervEspera := 0;
            ListaDesc[desc].FatiaTempo := 0;
            indice := 0;
            repeat indice := indice + 1;
               until TabProc1[Processador,indice] = 0;
            TabProc1[Processador,indice] := ListaDesc[
                                            desc].NomeProc;
            InsLDPont (PTR(@ProcProntos), desc, Pronto);
            msg[0] := 0;
            msg[1] := 12;
                              (* envia mensagem de inclu-
                                 sao *)
            msg[2] := Processador;
            msg[3] := ListaDesc[desc].NomeProc;
            writeln (msg[2], msg[3]);
            if (Processador + 1) MOD Pdores = 2
               then InsereMens(PTR(@Envia12), 0, 0,
                    PTR(@msg), 0, false, 4, erro)
               else InsereMens(PTR(@Envia14), 0, 0,
                    PTR(@msg), 0, false, 4, erro);
         end
         else erro := ExcessoProc;
      end;
   end;
   if filhos <> 0
      then begin
                              (* ja nao esta mais na inici-
                                 alizacao *)
      if (DescAtual <> 0) AND (pai = ProcAtual)
      then begin
         ListaDesc[DescAtual].Filhos := ListaDesc[
            DescAtual].Filhos + filhos;
         desc := RetLDPont (PTR(@ProcProntos));
         InsLPont (PTR(@BloqCria), DescAtual, BloqCriacao);
      end;
      Escalonador;
   end;
   Maquina;
end;


(*********************************************************)
procedure Termina;
var
   desc, auxiliar, buffer, descpai, pai, erro : INTEGER;
   msg : ARRAY [0..4] OF INTEGER;
begin
   writeln ('Termina');
   pai := ListaDesc[DescAtual].Pai;
   msg[0] := 0;
   msg[1] := 13;
   msg[2] := Processador;
   msg[3] := ProcAtual;
   msg[4] := pai;
   writeln (msg[2], msg[3], msg[4]);
   if (Processador + 1) MOD Pdores = 2
      then InsereMens (PTR(@Envia12), 0, 0, PTR(@msg), 0,
                       false, 5, erro)
      else InsereMens (PTR(@Envia14), 0, 0, PTR(@msg), 0,
```

```
                        false, 5, erro);
   auxiliar := 0;
   repeat auxiliar := auxiliar + 1
      until TabProc1[Processador,auxiliar] = ProcAtual;
   TabProc1[Processador,auxiliar] := 0;
   while ListaDesc[DescAtual].ListaMens.Inicio <> 0 do
   begin
      buffer := RetListaMens (DescAtual, 0);
      DevolveBuf(buffer);
      while AreaBuf[buffer].Cadeia <> 0 do
      begin
         buffer := AreaBuf[buffer].Cadeia;
         DevolveBuf (buffer);
      end;
   end;
   if (((pai LSR 13) + 1) = Processador) AND (pai <> 0)
      then AvisaPai (pai);
   desc := RetLDPont (PTR(@ProcProntos));
   InsLDPont (PTR(@DescVazios), DescAtual, 0);
   Escalonador;
   Maquina;
end;
```

APÊNDICE B

DEMONSTRAÇÃO DA MANIPULAÇÃO DA ÁREA COMPARTILHADA

A área compartilhada foi definida como um "buffer" circular na seção III.8, onde as mensagens são inseridas sequencialmente e retiradas na ordem em que foram colocadas. O algoritmo utilizado neste trabalho é uma modificação do algoritmo detalhado em MARKENSON (S.D.).

Suponhamos que (figura III.14):

. o "buffer" possui M elementos;

. o apontador P aponta para a posição vazia;

. o apontador C aponta para a posição cheia;

.. quando o "buffer está vazio temos, C=P;

. quando o "buffer" está cheio temos, C=P+1.

Os algoritmos de inserção e remoção ficam:

- inserção

```
begin
    if C=M then C:=1
           else C:=C+1;
    insere elemento
end
```

- remoção

```
begin
    if P=M then P:=1
           else P:=P+1;
    retira elemento
end
```

O teste para a condição de limite M do "buffer" pode ser simplificado diretamente para uma atribuição, utilizando a função MOD da linguagem PASCAL, ficando:

. para a inserção: C:=((C+1) MOD M);

. para a remoção: P:=((P+1) MOD M);

Esses algoritmos não levam em consideração a inserção ou remoção de mais de um elemento. Generalizando os algoritmos teremos:

- inserção

```
begin
    insere os elementos
    C:=( (C+ #elementos) MOD M);
end
```

- remoção

```
begin
    retira os elementos
    P:=( (P+ #elementos) MOD M);
end
```

Deve-se também cuidar para que elementos só sejam colocados no "buffer" enquanto houver espaço para tal. Desta forma é necessário um teste antes da inserção de elementos para verificar se o "buffer" possui espaço disponível para a mensagem:

- inserção

```
if C>P then livres:= M - (C-P)
       else livres:= P - C - 1;
if livres >= #elementos then
    begin
```

```
            insere os elementos
            C:=( (C+ #elementos) MOD M);
        end
        else não coloca elementos
```

Se o "buffer" estiver vazio a quantidade de espaços livres é igual ao tamanho total do "buffer" e P=C. Um teste a mais pode ser colocado no algoritmo acima para verificar esta condição.

De acordo com a manipulação descrita na seção III.7.3.2 é necessária a remoção de todos os elementos do "buffer". Um teste deve ser feito também durante a retirada para que, quando o "buffer" estiver vazio, acabe a remoção. Portanto, o algoritmo fica:

- remoção

```
    begin
        while P <> C do
            begin
                retira os elementos
                P:=( (P+ £elementos) MOD M);
            end
    end
```