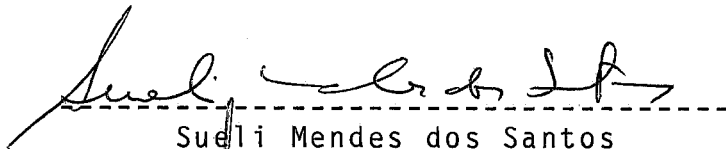


ESPECIFICAÇÃO DE UM NÚCLEO DE UM SISTEMA OPERACIONAL PARA
SUORTE DOS ATRIBUTOS DE CONCORRÊNCIA DA LINGUAGEM CHILL
EM SISTEMAS DISTRIBUÍDOS: UMA APLICAÇÃO EM TELEFONIA.

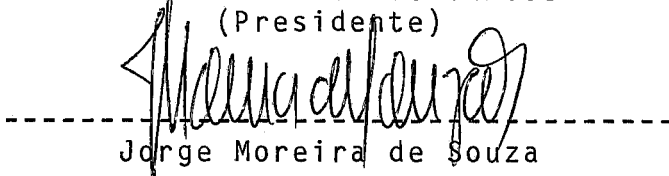
Carlos César Ferreira Araújo

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
OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

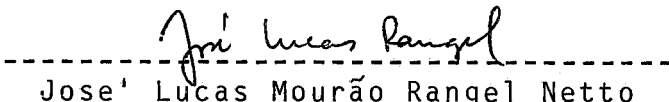
Aprovada por:



Sueli Mendes dos Santos
(Presidente)



Jorge Moreira de Souza



Jose' Lucas Mourão Rangel Netto

RIO DE JANEIRO, RJ - BRASIL

FEVEREIRO DE 1984

ARAÚJO, CARLOS CÉSAR FERREIRA

Especificação de um Núcleo de um Sistema Operacional para Suporte dos Atributos de Concorrência da Linguagem CHILL em Sistemas Distribuídos: Uma Aplicação em Telefonia (Rio de Janeiro) 1984.

VIII, 183 p, 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas, 1984).

Tese - UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, CPqD.

1. NÚCLEO CHILL.

I. COPPE/UFRJ II. Título (série)

Agradeço a direção do Centro de Pesquisa e Desenvolvimento da Telebrás - CPqD, pela oportunidade de realização do trabalho.

A minha orientadora Sueli Mendes dos Santos pela coordenação do trabalho.

Aos amigos Paulo Roberto Luz, Teodoro Pfeiffer e Sálvio Calicchio Sobrinho pelas contribuições técnicas.

As colegas Eloisa Quitério e Cristina Baraldi pela elaboração gráfica e digitação.

Agradeço especialmente a minha família pelo exemplo, educação e apoio e a minha esposa Marisa Ximenes Braga Araújo pelo incentivo, apoio e compreensão.

Agradeço também a todos os companheiros do Departamento de Comutação do CPqD que tornaram possível este trabalho.

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

ESPECIFICAÇÃO DE UM NÚCLEO DE UM SISTEMA OPERACIONAL PARA SUPORTE DOS ATRIBUTOS DE CONCORRÊNCIA DA LINGUAGEM CHILL EM SISTEMAS DISTRIBUÍDOS (UMA APLICAÇÃO 'A TELEFONIA)

CARLOS CÉSAR FERREIRA ARAÚJO

FEVEREIRO DE 1984

Orientador: Professora Sueli Mendes dos Santos

Programa: Engenharia de Sistemas

Este trabalho faz parte de um projeto desenvolvido no Centro de Pesquisa e Desenvolvimento da TELEBRAS (CPqD) junto ao Departamento de Comutação.

O trabalho consiste da especificação de uma arquitetura de multiprocessadores, do estudo de algumas linguagens com atributos de concorrência, da escolha de uma linguagem base a ser utilizada no desenvolvimento e aplicação, da especificação de um núcleo que suporta os atributos de concorrência da linguagem escolhida e de um sistema operacional (E/S, configuração, falhas, etc.) visando a aplicação dos mesmos em centrais telefônicas e de telex.

Os objetivos principais do trabalho são reunir bibliografia sobre os assuntos mencionados e fornecer uma contribuição para o desenvolvimento dos sistemas acima referidos, que pela complexidade e confiabilidade exigidas requerem contínuos estudos e aperfeiçoamento em sua elaboração.

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

SPECIFICATION OF AN OPERATIONAL SYSTEM'S NUCLEUS TO SUPPORT THE CONCURRENT ASPECTS OF THE LANGUAGE CHILL IN DISTRIBUTED SYSTEMS (AN APPLICATION TO TELEPHONY)

CARLOS CÉSAR FERREIRA ARAÚJO

FEBRUARY, 1984

Chairman: Sueli Mendes dos Santos

Department: Engenharia de Sistemas

This work is part of a project developed in the Switching Department of Telebra's Development and Research Center.

The work consists of a multiprocessor's architecture specification, the study of some languages with concurrent aspects, the choice of a basic language to be used in the development and application, the specification of a nucleus that supports the concurrent aspects of the basic language and the specification of an operational system (I/O, fails, configuration, etc.). All these things are to be used on Switching Systems.

The main objectives of this work are to get bibliography about the subjects mentioned above and to give a contribution to the development of Switching systems. These type of systems requires continuous studies and improvements on their elaboration.

ÍNDICE

CAPÍTULO I	- INTRODUÇÃO	1
I.1	- COMPUTAÇÃO	1
I.2	- TELEFONIA	3
I.3	- DESCRIÇÃO DOS PRÓXIMOS CAPÍTULOS	5
CAPÍTULO II	- MULTIPROCESSAMENTO E APRESENTAÇÃO DA ARQUITETURA DE MULTIPROCESSADORES	7
II.1	- INTRODUÇÃO	7
II.2	- SISTEMAS COM MULTIPROCESSAMENTO	7
II.2.1	- SISTEMAS DE MULTIPROCESSAMENTO EM REDE	7
II.2.2	- SISTEMAS DE MULTIPROCESSAMENTO LOCAL	8
II.2.2.1	- QUANTO AO PORTE DA UCP	9
II.2.2.2	- QUANTO À DISTRIBUIÇÃO DE FUNÇÕES	9
II.2.2.3	- QUANTO À LIGAÇÃO FÍSICA ENTRE OS PROCESSADORES	10
II.2.2.4	- QUANTO À DISTRIBUIÇÃO DE MEMÓRIA	11
II.2.2.5	- QUANTO À DISPOSIÇÃO DOS MÓDULOS	14
II.3	- ESCOLHA DA TOPOLOGIA "HARDWARE" (ARQUITETURA)	16
II.4	- ARQUITETURA DE MULTIPROCESSADORES BASE	16
II.4.1	- DESCRIÇÃO DA ARQUITETURA BASE	16
II.4.2	- DIAGRAMA DE BLOCOS DA ARQUITETURA COMIC	18
II.4.2.1	- SISTEMA	18
II.4.2.2	- MÓDULO DE COMUNICAÇÃO	20
II.4.3	- INTERFACE COM O "SOFTWARE" (PROCESSADOR LOCAL)	22
II.4.4	- COMUNICAÇÃO NA ARQUITETURA COMIC	25
II.4.5	- IMPLEMENTAÇÃO	27
CAPÍTULO III	- CONCORRÊNCIA E LINGUAGENS COM ATRIBUTOS DE CONCORRÊNCIA	28
III.1	- INTRODUÇÃO	28
III.2	- LINGUAGENS CONCORRENTES	29
III.2.1	- PASCAL CONCORRENTE	30
III.2.2	- MODULA	32
III.2.3	- ADA	34
III.2.4	- CHILL	39
III.2.5	- CONCLUSÃO	48
III.3	- RESUMO DAS LINGUAGENS APRESENTADAS QUANTO AOS SEUS ASPECTOS DE CONCORRÊNCIA	48

CAPÍTULO IV	- LINGUAGEM BASE E ATRIBUTOS DE CONCORRÊNCIA IMPLEMENTADOS	50
IV.1	- AVALIAÇÃO DAS LINGUAGENS APRESENTADAS QUANTO AOS SEUS ASPECTOS DE CONCORRÊNCIA	50
IV.2	- ESCOLHA DA LINGUAGEM BASE	55
IV.3	- APRESENTAÇÃO DOS ASPECTOS DE CONCORRÊNCIA A SEREM IMPLEMENTADOS	56
IV.3.1	- SUB-CONJUNTO DA LINGUAGEM CHILL	56
IV.3.2	- OUTROS ASPECTOS DE CONCORRÊNCIA A SEREM IMPLEMENTADOS	58
CAPÍTULO V	- ESTRUTURA DO NÚCLEO E SUAS FUNÇÕES	61
V.1	- FUNÇÕES DO NÚCLEO	61
V.1.1	- DIAGRAMA DE INTERAÇÃO DAS FUNÇÕES	63
V.1.2	- DESCRIÇÃO DAS FUNÇÕES	65
V.2	- ESTRUTURA DO NÚCLEO	72
V.3	- BLOCO DE COMUNICAÇÃO GLOBAL	73
V.4	- INTERFACES DO NÚCLEO	74
V.5	- ESTRUTURA DE UM PROGRAMA APLICATIVO	80
CAPÍTULO VI	- O NÚCLEO, SEUS ELEMENTOS DE CONTROLE E RECURSOS DE CONCORRÊNCIA	82
VI.1	- ELEMENTOS DE CONTROLE DO NÚCLEO	82
VI.2	- RECURSOS DE CONCORRÊNCIA	83
VI.2.1	- CONTROLE DE INICIAÇÃO	83
VI.2.1.1	- PRIMITIVAS	83
VI.2.1.2	- ÁREA DE COMUNICAÇÃO	84
VI.2.2	- GERÊNCIA DE PROCESSOS	84
VI.2.2.1	- DIAGRAMA DE ESTADOS DE UM PROCESSO	84
VI.2.2.2	- TIPOS DE PROCESSOS	88
VI.2.2.3	- PRIMITIVAS	88
VI.2.2.4	- ÁREA DE COMUNICAÇÃO	91
VI.2.2.5	- CÉLULA DESCRITORA DE PROCESSO	91
VI.2.2.6	- CÉLULA DE EXECUÇÃO CONDICIONAL	96
VI.2.2.7	- PILHAS	98
VI.2.3	- CONTROLE DE TEMPORIZAÇÃO	99
VI.2.3.1	- CLASSES DE TEMPORIZAÇÃO	99
VI.2.3.2	- TIPOS DE TEMPORIZAÇÃO	99
VI.2.3.3	- PRIMITIVAS	99
VI.2.3.4	- ÁREA DE COMUNICAÇÃO	100
VI.2.3.5	- CÉLULA DE TEMPORIZAÇÃO	101
VI.2.4	- CONTROLE DE SINAIS	103
VI.2.4.1	- CLASSES DE SINAIS	103
VI.2.4.2	- TIPOS DE SINAL	104
VI.2.4.3	- PRIMITIVAS	104
VI.2.4.4	- ÁREA DE COMUNICAÇÃO	107
VI.2.4.5	- CÉLULAS DE SINAIS	107

VI.2.5	- CONTROLE DE COMUNICAÇÃO EXTERNA	113
VI.2.6	- CONTROLE DE FALHAS	114
VI.2.6.1	- CLASSES DE FALHAS	114
VI.2.6.2	- TIPOS DE FALHAS	114
VI.2.6.3	- PRIMITIVAS	115
VI.2.6.4	- PROCESSO DE TRATAMENTO DE FALHAS	116
VI.2.6.5	- NÍVEL DE TRATAMENTO	117
VI.2.7	- CONTROLE DE INTERRUPTÕES	118
CAPÍTULO VII - CONCLUSÃO		121
VII.1	- RESULTADOS OBTIDOS	121
VII.2	- CONTINUAÇÃO DESTE TRABALHO	121
ANEXO I - DESCRIÇÃO DO NÚCLEO EM LINGUAGEM DE ALTO NÍVEL		122
I.1	- DESCRIÇÃO DO NÚCLEO	123
I.2	- DESCRIÇÃO DE UM PROGRAMA DO USUÁRIO	161
I.3	- BLOCO DE COMUNICAÇÃO GLOBAL	170
ANEXO II - DESCRIÇÃO DO SISTEMA OPERACIONAL		171
II.1	- CONFIGURADOR	173
II.2	- DEPURADOR	174
II.3	- GERÊNCIA DE RECURSOS SOFTWARE	176
II.4	- GERÊNCIA DE PERIFÉRICOS	177
II.5	- GERÊNCIA DE TEMPO	178
II.6	- GERÊNCIA DE FALHAS, ALARMES E TESTES	179
II.7	- GERÊNCIA DE RASTREAMENTO	179
BIBLIOGRAFIA -		181

CAPÍTULO I

INTRODUÇÃO

Neste capítulo é apresentada a evolução da computação, sua importância, assim como alguns conceitos que são referenciados ao longo da tese. Também é apresentada uma descrição sucinta dos demais capítulos.

I.1. COMPUTAÇÃO

A computação deve ser entendida como tudo que se relaciona com os computadores, ou seja, sua organização, arquitetura, componentes, programas, linguagens, aplicações e demais setores a eles ligados. A importância da computação pode ser medida pela importância que adquiriu o seu objeto de estudo: O computador. Anualmente enormes quantias são gastas no mundo inteiro no desenvolvimento, produção, manutenção e comercialização dos computadores e seus serviços. A economia de tempo e dinheiro por eles proporcionados os tornam essenciais para a sociedade, e a cada dia mais requisitados, podendo-se prever que em futuro bem próximo o computador será o nosso companheiro do dia a dia.

Tudo isto é proporcionado pela rápida evolução tecnológica experimentada por este setor da atividade humana. É possível afirmar que essa revolução que estamos presenciando é tão importante para a sociedade como o foi a revolução industrial do sec. XVIII, já que, como aquela, também provoca mudanças econômicas e sociais profundas na sociedade. O avanço tecnológico diversificou a aplicação dos computadores, da mesma maneira que estes tornaram realizáveis projetos anteriormente impossíveis do ponto de vista técnico. Hoje em dia os computadores são largamente utilizados em aplicações das mais variadas tais como: medicina, telecomunicações, educação, indústria, etc.

As técnicas de projeto, desenvolvimento, confecção e testes de circuitos eletrônicos, tem apresentado constantes e significativos avanços, porém é na parte de componentes eletrônicos que a evolução do "hardware" é mais sentida. Conceitos e técnicas são introduzidos e assimilados na mesma velocidade em que são substituídos por outros conceitos mais sofisticados. De um custo cada vez mais reduzido e tamanhos menores são produzidos circuitos com capacidade (número de funções) cada vez maior, menor potência consumida e maior velocidade. Estes componentes facilitam o projeto, simplificam o

circuito e diminuem o tempo de desenvolvimento, fatos que irão repercutir no custo e desempenho do produto. A partir das técnicas de produção dos circuitos integrados LSI (Integração em Larga Escala - 1970) surgiram os microprocessadores. Eles popularizaram os computadores tornando-os acessíveis ao público em geral e às mais variadas aplicações devido a sua simplicidade, baixo custo e grande capacidade.

As técnicas de projeto e programação de "software" tem apresentado avanços, não tão evidentes como no "hardware", mas de grande importância para a computação, já que as duas partes se completam. No início da computação pouca importância era dada ao "software", já que o custo dos circuitos ("hardware") era muito superior ao custo da programação ("software"). Hoje em dia esses papéis se inverteram, e devido ao alto custo do "software" em relação ao "hardware" muita atenção tem sido dada ao seu desenvolvimento.

Existem basicamente tres tipos de sistemas multiprogramados ou combinações desses tres tipos: Sistemas de lote, sistemas de tempo compartilhado e sistemas de tempo real.

Os sistemas de lote manuseiam um fluxo contínuo de entrada, processamento e saída em um único processador (SPOOLING). Estes sistemas são muito utilizados em centros de processamentos comerciais e centros acadêmicos, onde existe a necessidade de um grande número de programas terminados por unidade de tempo (throughput).

Os sistemas de tempo repartido (iterativo) são aqueles em que vários usuários interagem com o sistema simultaneamente através de terminais de vídeo, teletipos, etc. São sistemas utilizados em aplicações que exigem um tempo de resposta pequeno a uma solicitação (~1s), devendo parecer ao usuário que só ele se utiliza da máquina no momento de uma intervenção. É muito utilizado em sistemas de reserva de passagens, em bancos, em pesquisas, etc.

Os sistemas de tempo real são semelhantes aos sistemas de tempo repartido, no que diz respeito a multiplicidade de tarefas em execução simultânea no processador, porém diferem deles nos tempos de resposta. Os sistemas de tempo real exigem um tempo de resposta bem rígido de acordo com a aplicação em questão, além de troca de informações entre as várias tarefas em execução. São sistemas de aplicação específica tais como telefonia, controle de temperatura, etc.

Com a diminuição dos custos e tamanho dos circuitos e com a introdução dos componentes com alta densidade de funções

(microprocessadores) surgiram os sistemas distribuídos. O sistema distribuído é formado por um conjunto de processadores que executam tarefas em paralelo e cooperam entre si para a execução de uma determinada função. Estes sistemas aliam a capacidade de multiplexação dos recursos de um processador (multiprogramação) a capacidade de execução de tarefas em paralelo de vários processadores (multiprocessamento). Este é o tipo de sistema apresentado na tese.

Outra parte do software que teve um grande avanço foram as linguagens de alto nível para programação. Elas tornaram viáveis muitas aplicações do computador devido a diminuição dos custos e das facilidades técnicas que apresentam. As linguagens de alto nível surgiram a partir do final da década de 50. FORTRAN, ALGOL, COBOL e PASCAL são alguns exemplos de linguagens de alto nível. Entre as facilidades oferecidas pelas linguagens de alto nível podemos citar:

- Maior eficiência, clareza e correção na programação (menor tempo e custo de desenvolvimento).
- Menor tempo de aprendizado (mais amigável).
- Menor tempo de depuração.
- Melhor documentação.
- Maior independência da máquina (o programador não necessita conhecer a máquina a fundo).
- Portabilidade dos programas.

Últimamente tem sido implementadas ferramentas para controle de concorrência nas linguagens de alto nível, visando sobretudo a uma maior eficiência na utilização dos recursos da máquina, além de facilitar e dar maior segurança na programação de sistema baseadas em multiprogramação e/ou multiprocessamento. Pascal Concorrente, ADA, Modula e CHILL estão entre as linguagens mais conhecidas com aspectos de concorrência.

I.2. TELEFONIA

A telefonia, como qualquer outro campo de atividade, foi altamente afetada pela introdução e evolução tecnológica dos computadores. Os sistemas telefônicos, no início totalmente mecânicos, estão hoje em dia se utilizando largamente dos computadores. A partir da década de 60 com as facilidades possibilitadas pelos computadores e seu custo cada vez menor, foram introduzidas funções cujo controle ficavam a cargo de

computadores. Outros fatores que tornam necessária a utilização dos computadores em centrais telefônicas são as novas facilidades e recursos oferecidos ao usuários e às operadoras.

Hoje em dia as centrais telefônicas destinam-se não somente a transmissão de voz, mas também a transmissão e armazenamento de dados (vídeo-texto), transmissão de informações confidenciais (saldo bancário), necessitando para isso de um sistema de controle rápido e seguro. As operadoras são contempladas com facilidades de operação e manutenção tais como bloqueio de assinantes por comando, mudança de categoria de assinantes, controle e leitura de taxaço automática, etc.

As centrais telefônicas precisam atender a requisitos tais como:

- alta confiabilidade
- baixo custo de instalação, operação e manutenção
- facilidade de expansão
- facilidade para implementação de novos serviços (conferência, discagem abreviada, siga-me, etc.) e inserção de novas tecnologias
- compatibilidade com padrões internacionais.

As primeiras centrais eletromecânicas possuíam controle bem distribuído, de modo que a falha em uma parte pouco afetava a central. Desta forma essas centrais possuíam alta confiabilidade, porém não eram de fácil modificação e tinham poucos serviços.

Com a utilização do computador em centrais telefônicas algumas funções foram centralizadas sob controle deste elemento. Embora isto facilitasse a operação, inclusão de novos serviços e adaptação de novas tecnologias, baixou a confiabilidade já que a falha no computador afetava todas as funções sob seu controle.

Com a redução dos custos e complexidade dos computadores, principalmente com o surgimento dos micros, estes passaram a ser utilizados em todas as fases do processo telefônico. Surgem dessa forma as centrais telefônicas baseadas em sistemas distribuídos (micros). Este tipo de sistema atende às necessidades de confiabilidade e às facilidades inerentes do computador. Estes tipos de sistema necessitam ser modular e implementam aspectos de concorrência e paralelismo devido a multiplicidade de tarefas em paralelo, alto grau de solicitação, etc.

Uma medida que demonstra o impacto e importância dos computadores nas telecomunicações, incluindo a telefonia, foi a criação de grupos de estudos e normas que tratam desse assunto pelo C.C.I.T.T. (orgão internacional de controle das

telecomunicações). Esses grupos introduziram mecanismos, normas e até linguagens para a descrição e o desenvolvimento de sistemas telefônicos baseados em computadores. As linguagens SDL e CHILL e estudos sobre sistemas distribuídos são algumas das contribuições mais recentes da aplicação dos computadores na telefonia.

I.3. DESCRIÇÃO DOS PRÓXIMOS CAPÍTULOS

CAPÍTULO II - Multiprocessamento e Apresentação da Arquitetura de Multiprocessadores.

Apresenta o conceito de sistema distribuído, discute os vários tipos de arquiteturas de multiprocessadores existentes e propõe as linhas gerais da arquitetura a ser utilizada como base para a tese.

CAPÍTULO III- Concorrência e Linguagens com Atributos de Concorrência.

Apresenta o conceito de concorrência, descreve e discute algumas linguagens com atributos de concorrência (ADA, Pascal concorrente, modula e CHILL).

CAPÍTULO IV - Linguagem Base e Atributos de Concorrência Implementados.

Apresenta a linguagem escolhida como base e os motivos dessa escolha. Descreve os atributos de concorrência implementados na tese.

CAPÍTULO V - Estrutura do Núcleo e suas Funções

Apresenta a estrutura do Núcleo e as suas diversas funções e seu relacionamento interno e externo.

Apresenta também a estrutura de um programa aplicativo e as interfaces do núcleo.

CAPÍTULO VI - O Núcleo, seus Elementos de Controle e Recursos de Concorrência.

Descreve e especifica os recursos de concorrência implementados e como é realizado o seu controle pelo núcleo.

CAPÍTULO VII- Conclusão.

Apresenta os resultados atingidos pelo trabalho e a sua possível continuação.

- ANEXO I - Descrição do Núcleo em Linguagem de Alto Nível.
Apresenta a estrutura e os algoritmos das funções do núcleo descritas em linguagem de alto nível.
Apresenta o corpo de um programa aplicativo descrito em linguagem de alto nível.
- ANEXO II - Descrição do Sistema Operacional.
Descreve as funções e estrutura do Sistema Operacional.
(E/S, configuração, falhas, calendário, etc).

CAPÍTULO II

MULTIPROCESSAMENTO E APRESENTAÇÃO DA ARQUITETURA

DE MULTIPROCESSADORES

II.1. INTRODUÇÃO

Verificamos no Capítulo I a evolução da eletrônica e a conseqüente importância assumida pelos computadores nos diversos campos de atividades humanas, notadamente nas telecomunicações, objeto de nosso interesse. Essa evolução se processou tanto em termos de "Hardware" como em "Software", pois há necessidade de um acompanhamento evolutivo entre os dois. Em "Hardware" houve o surgimento de sistemas com multiprocessamento (Capítulo I), ou seja, vários processadores executando tarefas paralelas e/ou compartilhando recursos. O paralelismo e o compartilhamento são algumas das vantagens do multiprocessamento sobre os sistemas comuns (um processador). Em "Software" houve o surgimento de linguagens de programação concorrente e protocolos de comunicação idealizados para lidar com o paralelismo e compartilhamento de recursos existentes nos sistemas de multiprocessamento. Nos deteremos, por hora, nos aspectos ligados ao "Hardware" deixando o "Software" para ser coberto com mais detalhes no próximo capítulo.

II.2. SISTEMAS COM MULTIPROCESSAMENTO

O custo dos processadores e a velocidade e capacidade cada vez maiores dos mesmos sugerem a distribuição das funções, antes realizadas por um processador, por vários processadores. Estes sistemas se tornam cada vez mais populares devido a gama variada de aplicações que podem atingir.

Os sistemas com multiprocessamento podem ser divididos basicamente em dois tipos: fracamente ligados - sistemas de multiprocessamento em rede e fortemente ligados - sistemas de multiprocessamento local. É feita a seguir uma breve descrição do sistema em rede (o sistema local será descrito com mais detalhes, já que se trata do sistema a ser empregado no projeto).

II.2.1. Sistemas de Multiprocessamento em Rede

Estes sistemas também são conhecidos como redes de computadores. São em geral constituídos por vários computadores independentes em termos físico e lógico, separados por distâncias relativamente grandes e que trocam informações que possibilitam a um utilizar os recursos de "Software" contidos no outro. As informações podem ser trocadas de várias maneiras: comutação de pacotes, troca de mensagens, etc.

Temos vários tipos de topologia de redes tais como: estrela, anel, etc.

Alguns autores propõem uma divisão deste tipo de sistema em dois: rede local e rede remota. As principais diferenças entre os dois são a distância entre os computadores e a velocidade de transmissão.

	DISTÂNCIA	VELOC. TRANS.
Rede Local	$d < 10 \text{ Km}$	$0,1\text{Mbps} < V < 10\text{Mbps}$
Rede Remota	$d > 10 \text{ Km}$	$V < 0,1\text{Mbps}$

Rede remota também pode ser entendida como sendo aquela onde vários computadores (médios e grandes em geral) antes totalmente independentes são interligados de modo a propiciar a troca de recursos de software. As redes remotas podem ser aplicadas em sistemas bancários, em sistemas de passagens aéreas, etc. (banco de dados). Exemplos de redes: ARPANET, ETHERNET, etc.

A rede local é um híbrido da rede remota e dos sistemas de multiprocessamento local, que tenta aproveitar a capacidade de compartilhamento de recursos do primeiro com o paralelismo do segundo. Devido a aplicação, essas redes empregam em geral computadores de médio porte (minis) interligando-se por meio de canais seriais especiais ou utilizando a rede telefônica (telecomunicações) com possibilidades técnicas disponíveis. Podemos ter tres tipos de redes no que diz respeito à função hierárquica dos componentes: centralizado, parcialmente distribuído e totalmente distribuído. No centralizado um computador da rede centraliza as funções de coordenação numa relação mestre escravo com os demais. Nos sistemas parcialmente distribuídos certas funções de coordenação são regionalizadas, ou seja, alguns computadores possuem funções específicas (comunicação) dentro da rede. No sistema totalmente distribuído não há nenhum computador com função específica para com a rede, nem hierarquia fixa definida (ETHERNET).

II.2.2. Multiprocessamento Local

O sistema de multiprocessamento local é constituído de vários processadores interligados, executando procedimentos paralelos e comunicando-se entre si a fim de realizar a função para a qual o sistema é destinado. Neste caso os processadores não são independentes um do outro e a principal característica do sistema é a alta capacidade de execução de tarefas em paralelo. É necessário o conjunto de todos os processadores para realizar a função. Este tipo de sistema geralmente é destinado a aplicações específicas tais como, controle de processos (telefonía), sistemas de desenvolvimento, etc.

Um sistema de multiprocessamento local pode ser classificado quanto à sua topologia "Hardware" a partir de cinco aspectos (também conhecida como a arquitetura de processadores).

- Quanto ao porte da UCP
- Quanto à distribuição de funções
- Quanto à ligação física
- Quanto à distribuição de memória
- Quanto à disposição dos módulos

II.2.2.1. Quanto ao Porte da UCP

Este aspecto define geralmente a aplicação do sistema. Devido ao fato dos microprocessadores estarem a cada dia mais baratos, terem a cada dia mais circuitos por área física (LSI, VLSI) que as demais tecnologias (bit slice, etc) e terem a cada dia mais poder (endereçamento, E/S, conjunto de instruções, rapidez, etc), eles estão atingindo um número cada vez maior de aplicações. Isto leva à utilização do microprocessador no projeto a ser definido.

II.2.2.2. Quanto à Distribuição de Funções

Temos tres tipos no que diz respeito à distribuição de funções: centralizado, parcialmente distribuído e totalmente distribuído.

a) Centralizado

O sistema centralizado possui um processador que controla os demais processadores numa relação mestre-escravo. Esse controle é feito sobre funções específicas e necessárias para o funcionamento do sistema tal como comunicação entre os processadores. Neste tipo, o processador central tem a maior hierarquia e pode ser duplicado para efeitos de confiabilidade, sendo que a duplicata só entra em ação se o processador original falhar (duplicação do tipo

escrava).

b) Parcialmente Distribuído

No sistema parcialmente distribuído existem alguns processadores realizando funções específicas para o funcionamento do mesmo, como por exemplo, a comunicação entre os processadores. Pode-se usar, para efeitos de confiabilidade, redundância para estes processadores com funções específicas.

c) Totalmente Distribuído

No sistema totalmente distribuído nenhum dos processadores que participam do sistema possui função específica para o funcionamento do mesmo. Uma falha em qualquer um dos processadores não afeta o funcionamento dos demais quanto ao aspecto "Hardware". Outro processador, dependendo da aplicação, pode assumir as funções do que falhou. Não há hierarquia fixa ou pre-estabelecida.

Em uma análise dos tres tipos, podemos verificar que o último e' o que apresenta as melhores características, embora seja o de mais difícil implementação. O sistema totalmente distribuído apresenta maior confiabilidade, maior modularidade e maior capacidade de expansão. Em termos de "hardware" estes aspectos vêem dos seguintes fatos: a falha em um processador não afeta o funcionamento do restante do sistema, os processadores são idênticos, facilitando a expansão e aumentando a modularidade.

II.2.2.3. Quanto à Ligação Física entre os Processadores

Temos, em geral, dois tipos principais de ligação física entre os processadores de um sistema de multiprocessamento local: ligação c/ canal serial e ligação em barra paralela.

a) Ligação com canal serial

O canal serial permite que os processadores do sistema fiquem a uma distância relativamente maior um do outro que no outro tipo de ligação (entre 10m e 100m em geral). Ha' uma redução na capacidade de transmissão (n. bits/tempo) em relação ao outro tipo de ligação, ja' que nesse tipo de ligação cada transmissão equivale a um bit de informação transmitida. E' mais simples de implementar fisicamente (1 linha serial). Possui sistemas e circuitos padrões para implementá-lo tipo RS 232, loop de corrente, MIL STD 1553, etc. Tem taxas de transmissão que vão desde 100 Hz ate' 2MHz. São geralmente assíncronas, o que faz

baixar as taxas de transmissão. Os casos de taxa mais alta já são síncronos (enviam o clock junto com os dados).

b) Ligação com barra paralela

A barra paralela aumenta a capacidade de transmissão, já que vários bits podem ser transmitidos em paralelo a cada transmissão em grande velocidade. A transmissão nesse caso é síncrona. Nesse caso consegue-se taxas de até ~ 10MHz de bits paralelos (o número de bits depende da palavra do processador).

Podemos verificar que é a aplicação a ser dada ao sistema que determina a ligação a ser usada. Para sistemas que não necessitem de uma distância física grande entre os processadores e necessitem de uma maior capacidade de transmissão, é utilizada a barra paralela (este é o caso da nossa aplicação).

II.2.2.4. Quanto à Distribuição da Memória

Podemos ter quatro tipos de distribuição de memória: memória comum compartilhada, memória distribuída, memória compartilhada-distribuída independente e memória totalmente distribuída.

a) Memória comum compartilhada

Neste caso todos os processadores se utilizam de uma memória comum para troca de informações. (A memória pode ser toda ou uma parte da memória do processador). Este tipo é muito utilizado em co-processamento. Tem como inconveniente o fato da memória se apresentar como um "gargalo", que restringe a capacidade de troca de informações e cuja falha provoca o não funcionamento de todo o sistema. Além do inconveniente de falha há também a queda da performance devido ao número de processadores que compartilham essa memória. No caso de memória totalmente compartilhada a performance é igual a $1/n$ onde $n = \text{no. de processadores}$. Figura (II.1).

b) Memória compartilhada distribuída

Cada processador possui uma parte de sua memória acessada, para efeitos de comunicação, por ele e pelos demais processadores do sistema. Possui maior modularidade que o tipo anterior, confiabilidade, pois a falha em um processador (módulo) não afeta os demais, além de aumentar a capacidade de comunicação. Tem como

um possível inconveniente, dependendo da aplicação, o fato de um processador local ser interrompido ao ter sua memória compartilhada acessada por outro. Superior ao anterior em termos de performance. Figura (II.2).

c) Memória compartilhada distribuída e independente

Cada processador possui uma memória local e uma memória, para comunicação. Esta memória para comunicação está distribuída em todos os processadores e possui acessos independentes, ou seja, pode ser acessada pelo processador local ou pelos demais processadores, sem que seja necessário interromper de alguma forma o processador local. Tem como vantagem o fato da comunicação não exigir controle desnecessário do processador local. Apresenta porém, maior complexidade de implementação que o tipo anterior. Figura (II.3).

Podemos concluir que o tipo C apresenta uma complexidade de implementação que só seria justificada através da impossibilidade do tipo B atender os requisitos da aplicação a ser dada ao sistema.

d) Memória totalmente distribuída

A comunicação é feita por enlace. A memória só é acessada pelo processador local. Este tipo ocupa a UCP durante a transmissão desnecessariamente, além de aumentar o tempo de transmissão de uma mensagem em relação aos outros tipos. Figura (II.4).

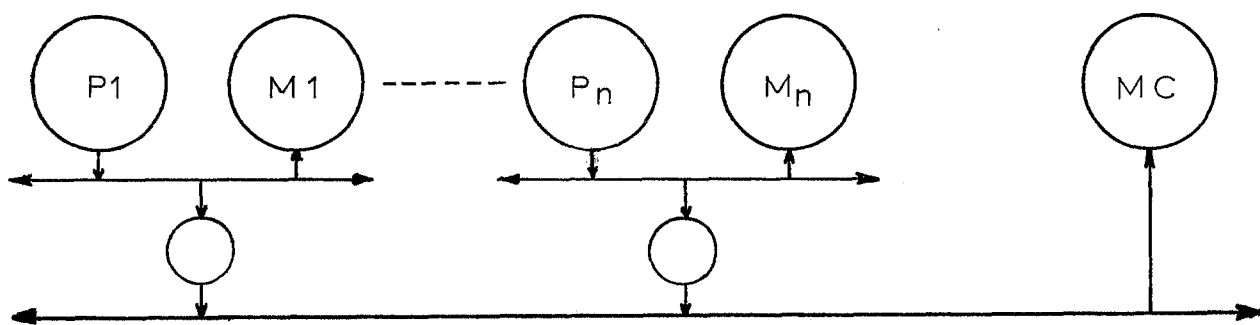


FIGURA (II.1) - MEMÓRIA COMUM COMPARTILHADA

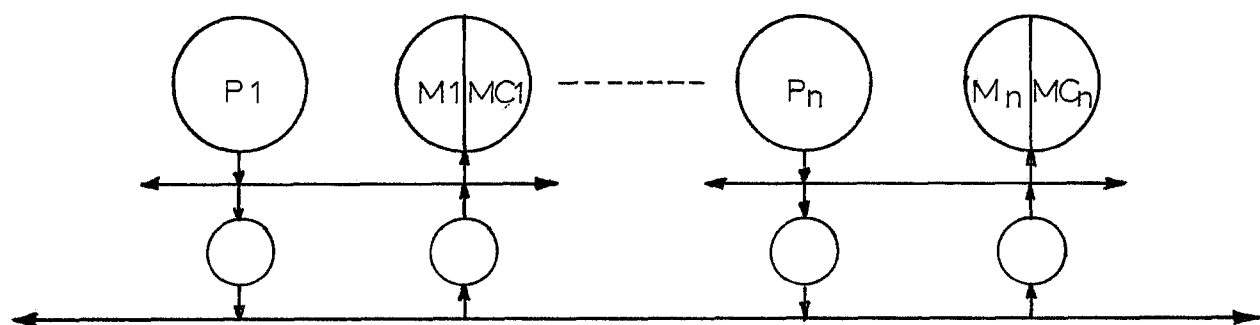


FIGURA (II.2) - MEMÓRIA COMPARTILHADA DISTRIBUÍDA

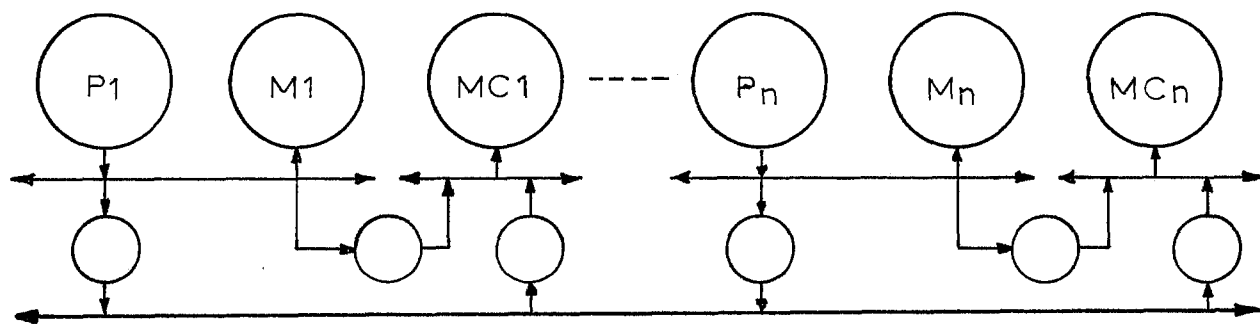


FIGURA (II.3) - MEMÓRIA COMPARTILHADA DISTRIBUÍDA E INDEPENDENTE

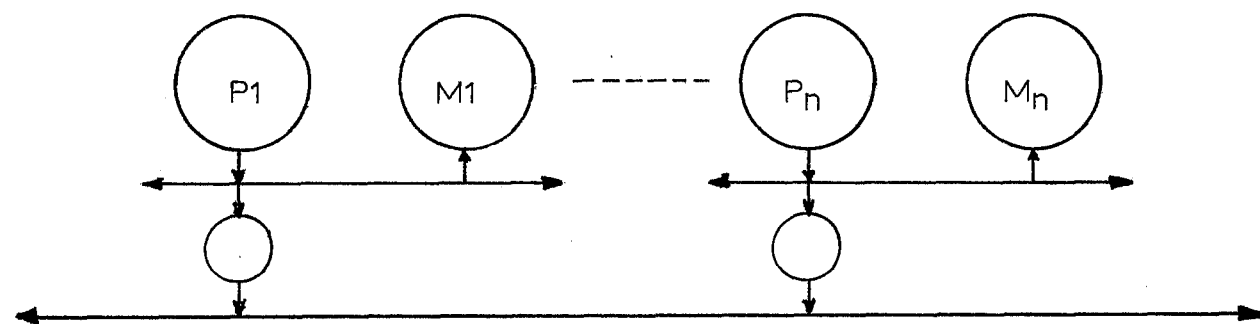


FIGURA (II.4) - MEMORIA TOTALMENTE DISTRIBUIDA

M - MEMORIA

MC - MEMORIA COMPARTILHADA

P - PROCESSADOR

II.2.2.5. Quanto à Disposição dos Módulos

Existe tres tipos básicos: anel, estrela e barra.

a) Anel

Neste tipo os módulos são dispostos em linha, interligando-se um a um ate' fechar o círculo. Figura (II.5). Este tipo de sistema e' simples , mas tem como inconveniente o fato da queda de um módulo afetar as comunicações da rede. Mecanismos devem ser implementados para evitar este fato.

b) Estrela

O sistema com ligação em estrela permite que cada processador seja ligado por meio de enlaces diferentes a mais de um processador. Devido a este fato este sistema possui grande flexibilidade, mas e' de implementação cara e difícil, devendo sua adoção ser justificada pelo tipo de utilização. Figura (II.6).

c) Barra

O sistema em barra interliga os vários módulos por meio de um enlace comum, que pode ser duplicado para efeitos de confiabilidade. Neste tipo de sistema a queda de um módulo não afeta os demais. A velocidade depende do tipo de barra utilizada (serial ou paralela). Devido a sua simplicidade e eficiência atingida e' o tipo de sistema mais empregado em multiprocessamento local. Figura (II.7).

O sistema em barra e' o utilizado neste trabalho, devido a sua simplicidade, capacidade de comunicação, confiabilidade e facilidade de expansão.

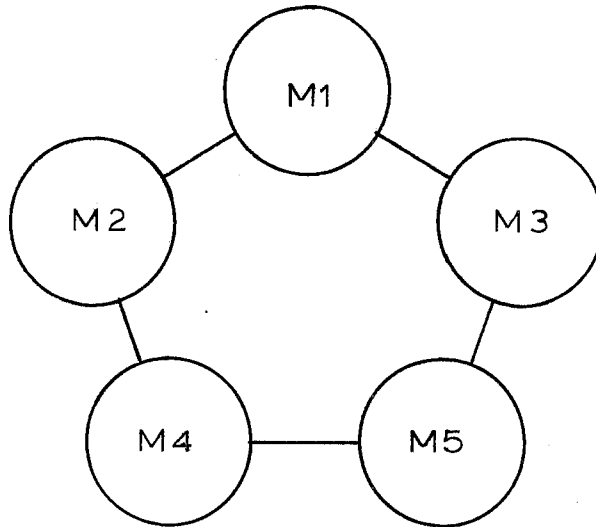


FIGURA (II.5) _ ANEL

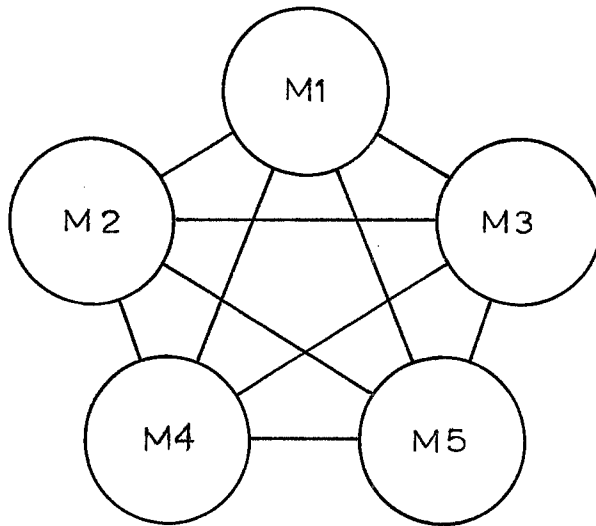


FIGURA (II.6) _ ESTRELA

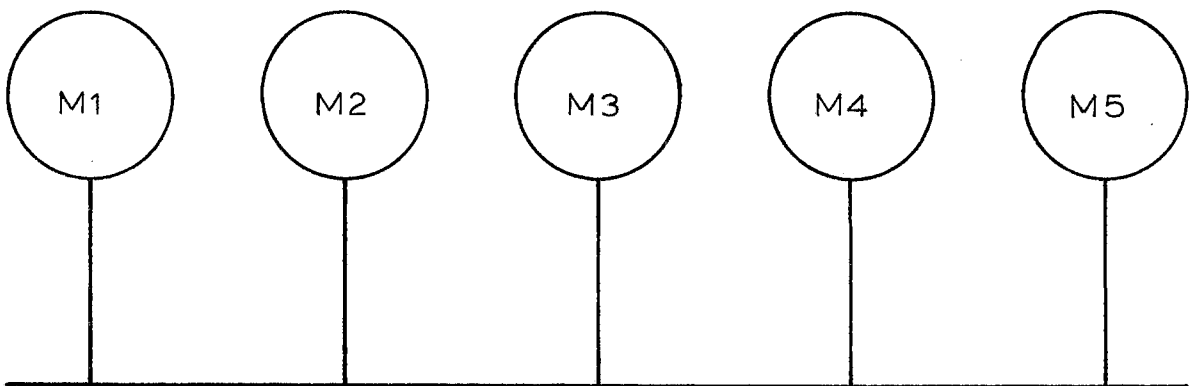


FIGURA (II.7) _ BARRA

M_ MÓDULO

II.3. ESCOLHA DA TOPOLOGIA HARDWARE (ARQUITETURA)

A aplicação a ser dada ao sistema tem importância fundamental na escolha da topologia hardware a ser empregada em um projeto. Para a maioria das aplicações em tempo real, incluindo a nossa (telecomunicações), o sistema de multiprocessamento baseado em micros, com ligação em barra paralela, memória compartilhada distribuída e com módulos totalmente distribuídos e' a que atende ao maior número de requisitos sem complicar em demasiado a implementação.

Este tipo de sistema e' modular, facilitando expansões, não possui "gargalos", possui maior confiabilidade, pois a falha em um módulo não afeta o funcionamento do resto do sistema, permite fácil reconfiguração pois, qualquer módulo pode, dependendo da aplicação, tomar as funções de outro, permite redundância e não possui hierarquia pre'-definida ou fixa. Além disso, podemos ainda destacar a alta capacidade de paralelismo permitida neste tipo de sistema .

II.4. ARQUITETURA DE MULTIPROCESSADORES BASE

II.4.1. Descrição da Arquitetura Base

A Arquitetura COMIC foi definida e projetada em estudos visando obtenção de um sistema de multiprocessamento local, a ser empregado principalmente na área de telecomunicações e que atendesse a requisitos pré-definidos. E' uma arquitetura baseada em micros, totalmente distribuída com comunicação através de barra paralela e com memória distribuída. Seu projeto possibilita a utilização de qualquer micro na sua implementação, porém para efeitos práticos o estudo e' realizado sobre o 8085 devido a sua simplicidade, capacidade e larga utilização em nosso País.

Foram estabelecidos alguns requisitos como metas a serem atingidas no projeto da arquitetura. Estes requisitos são:

1. Liberar micros de controle da comunicação.
2. Não possuir módulos com funções específicas para o funcionamento do sistema ("Hardware").
3. Evitar "gargalos".
4. Facilitar reconfiguração.
5. Facilitar expansão.

6. Alta confiabilidade.
7. Boa capacidade de comunicação.
8. Não ter hierarquia pré-definida e/ou fixa.

O primeiro requisito foi atingido através da utilização de um controle microprogramado, existente em cada módulo, para cuidar da comunicação entre processadores e agilizar a comunicação. Este controle está associado a uma lógica de DMA responsável pelo acesso e proteção a acesso indevido na parte da memória local utilizada para a comunicação.

O segundo requisito foi realizado através da modularização. Todos os módulos básicos são idênticos no que diz respeito ao hardware e desta maneira as funções são totalmente distribuídas no sistema.

Através da alta modularidade evita-se gargalos e atinge-se o requisito tres. Não há circuito que tenha função específica para o funcionamento da rede e que não tenha sido totalmente distribuído pelos módulos. A barra paralela que poderia ser considerada como gargalo, embora não seja circuito (pequena probabilidade de falha), foi n-uplicada para evitar esse problema (será apenas duplicada para efeitos de implementação).

A modularização e consequente distribuição total de funções de controle para o funcionamento da rede permite a realização do requisito quatro. Uma vez que todos os módulos básicos são idênticos e não possuem função específica para o funcionamento da arquitetura, qualquer módulo falho pode ter suas funções assumidas por outro dependendo da aplicação.

O quinto requisito também foi atingido através da modularização. A expansão não necessita de circuito adicional nem modificações no sistema existente, sendo necessário apenas a introdução do novo módulo. A decodificação de processador destino na barra também permite que a expansão seja feita até os limites das restrições físicas.

Para atingir o requisito seis houve uma total distribuição no sistema eliminando a existência de gargalos e circuitos especiais e únicos para controle da arquitetura. A utilização da DMA para proteção de acesso a memória local aumenta a confiabilidade. A falha em um módulo, seja a falha no circuito de comunicação ou no processador não afeta a barra ou o resto do sistema. A duplicação da barra também visa esse requisito.

O requisito sete foi realizado pela multiplexação da barra paralela, pela utilização de uma barra paralela rápida com sinais próprios, pela utilização do circuito específico com DMA para controle da comunicação (liberando a CPU), pela introdução de sinais de tamanho variável (diminuir a carga de dados transferidos) e pela utilização do conceito de "difusão" (que serão definidos posteriormente), agilizando a comunicação. A DMA possui canais independentes para transmissão e recepção de sinais.

Finalmente o oitavo requisito foi atingido através da utilização de um circuito de prioridade rotativa, ligado ao controle de comunicação. A prioridade é estabelecida num sentido e a partir do funcionamento do sistema ela passa a rodar, dando a maior prioridade a um módulo de cada vez.

II.4.2. Diagrama de Blocos da Arquitetura COMIC

II.4.2.1. Sistema

O sistema é constituído de N módulos interligados por M barras paralelas externas (para implementação serão utilizados 63 módulos e duas barras). Figuras (II.8) e (II.9).

Cada módulo pode ser subdividido em tres partes

O módulo local aplicativo define as características de aplicação desse módulo e sua interface com o mundo externo, tais como entrada/saída, co-processadores, etc. O módulo local básico é constituído do processador e da memória. O terceiro módulo é responsável pela comunicação com os demais processadores. A união do módulo local básico e módulo de comunicação é chamado módulo básico e tem como característica o fato de ser idêntico para toda a rede, logo as características que diferem nos módulos estão no módulo local aplicativo. Do ponto de vista local tem-se um processador com a configuração que for mais aplicada ao problema.

A barra interna é a via de ligação entre os módulos, sendo independente da barra externa e estando separado desta pelo módulo de comunicação.

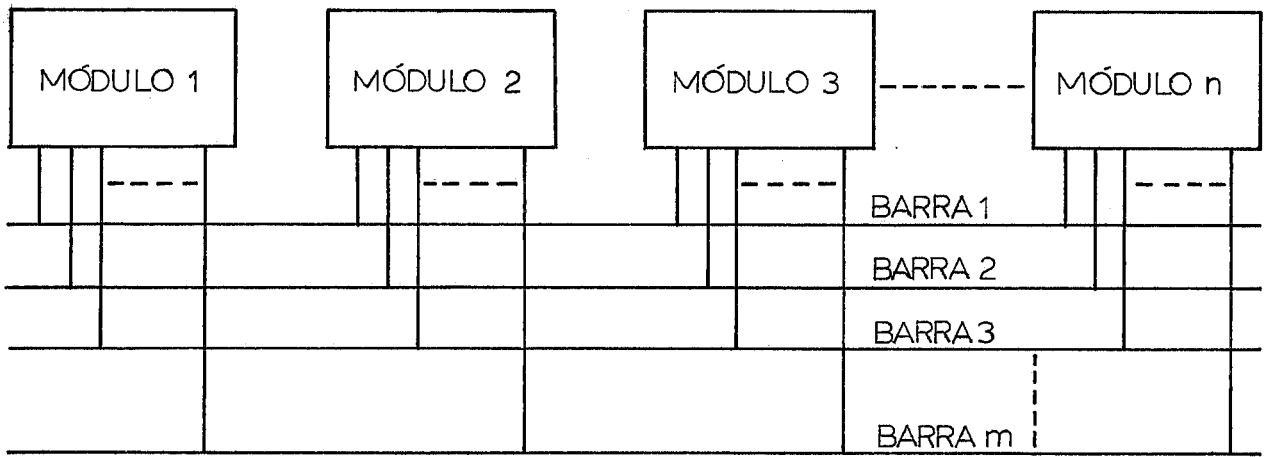


FIGURA (II.8)

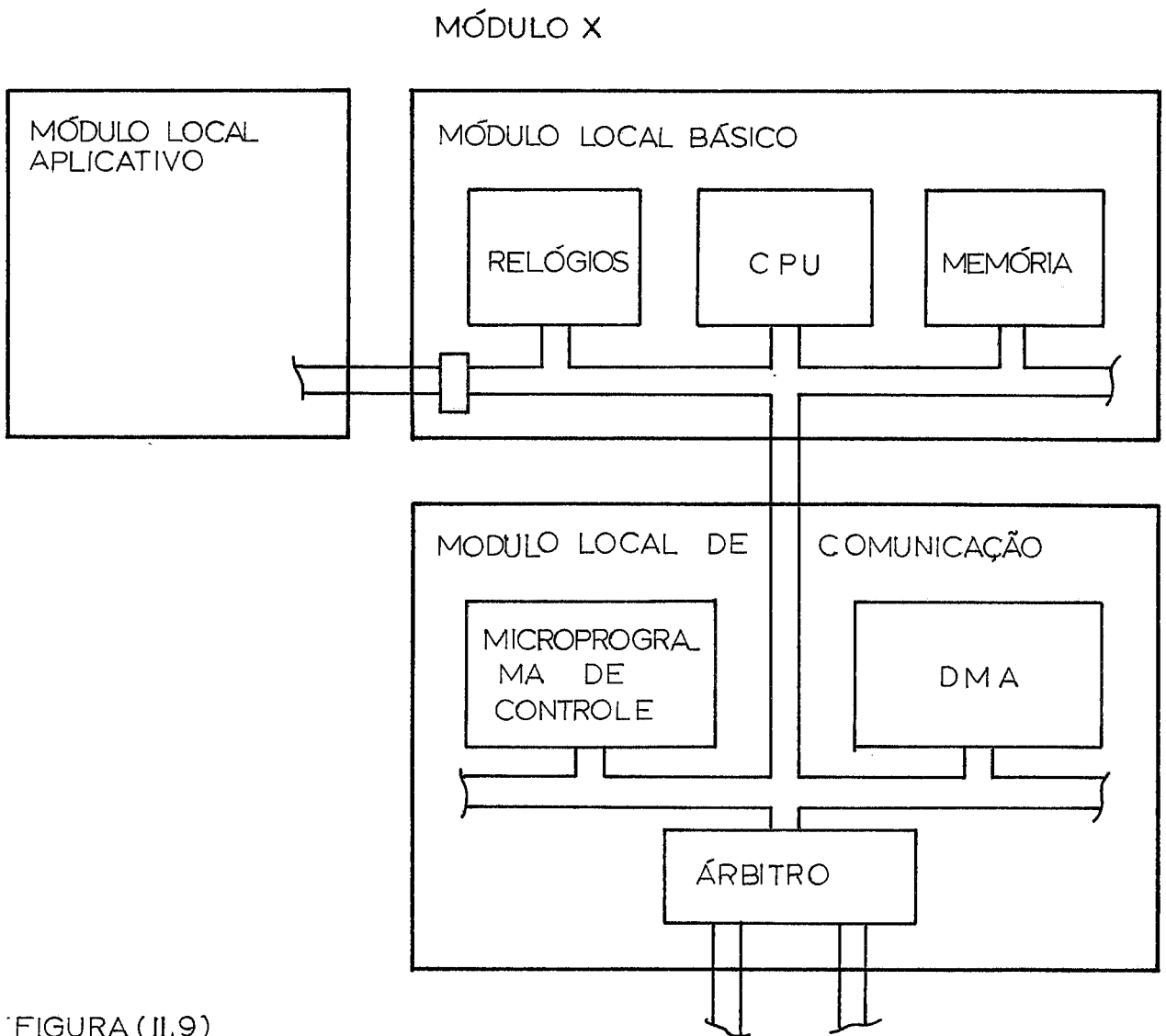


FIGURA (II.9)

II.4.2.2. Módulo de Comunicação

O módulo de comunicação é um sistema "hardware" que prove todo o controle para comunicação de processadores ligados a uma barra paralela. Basicamente o módulo de comunicação é constituído de um circuito de controle microprogramado, um circuito de DMA e registros de uso geral. Figura (II.10).

Diagrama do Módulo de Comunicação

A função de cada unidade apresentada no diagrama é explicada a seguir:

a) Registros de dados

de saída - adquire o dado a ser transferido da DMA e armazena-o até que o sistema da barra o transfira para o destino.

de entrada - adquire da barra (conforme a selecionada) o dado enviado pelo processador origem.

b) Multiplex de entrada

Acionado pelo microcontrole, permite a passagem dos dados conforme a barra selecionada.

c) Registro de endereço de destino

O processador local escreve nesse registro o endereço do processador destino. Temos um endereço para difusão e um endereço para cada processador do sistema.

d) Registro de operação

O processador local utiliza esse registro para informar ao controle a ação desejada no caso de envio ou recepção de mensagens. Podemos ter as seguintes opções:

Permite entrada

Inicia transmissão

Nohold

Verificação das vias

Auto diagnóstico

e) Controle de DMA

Provê o controle de acesso direto à memória local. Contem registros internos que devem ser programados pelo processador local de acordo com a necessidade. Interage com o registro de dado para receber ou transmitir para a memória. E' sincronizado pelo controle microprogramado.

f) Registro de estado

Informa ao processador local as condições do controle e das barras, estão também nesse registro o resultado da operação precedente. Deve ser lido sempre que houver uma interrupção para o processador local.

g) Interrupção

Aciona a interrupção para o processamento local indicando fim de operação ou erro.

h) Controle microprogramado

Faz o sequenciamento dos sinais de controle a partir de sinais vindos da barra, das condições do registro de operações e da DMA. Responsável pelo controle da comunicação.

i) Circuito de tempo

Fornece os tempos necessários para evitar espera indefinida do controlador. Sua ação provoca o acionamento de condições de erro.

j) Circuito de prioridade de acesso à barra

Provê um esquema para resolver os problemas de colisão no acesso à barra. A prioridade e' rotativa.

l) Decodif. de entrada (Proc.)

Interpreta o endereço do processador destino colocado na barra pelo outro. Sinaliza para o microprograma se for ele o solicitado.

m) Decodif. de sinais

E' acionado pelo micro-controle, coloca na barra e no resto do circuito os estados da comunicação (início de transf., fim de transf.) e outras informações necessárias para uma comunicação.

II.4.3. Interface com o Software (Processador Local)

Existem registros que necessitam ser programados pelo processador local para realização de alguma operação e outros que podem ser lidos para que o processador local tome conhecimento da condição do controlador de comunicação. Estes registros fornecem flexibilidade na implementação do sistema.

a) Registro de estado do controle

Indica a situação do controle. Podemos ter as seguintes condições: ocupado, transferência OK, erro, transmissão/recepção, erro na barra, "time out" de sinal, difusão. O sinal erro e' um "ou" dos eventos que podem ocasionar uma falha no funcionamento do controle.

b) Registro de endereço do processador destino

Indica em n bits $2n$ endereços de processadores de destinos. Um endereço e' utilizado para difusão e os demais para endereço efetivo de processador.

c) Registro de operação

Indica para o controlador como este deve proceder no caso de recepção de sinal ou que operação o controlador deve executar. Podemos ter as seguintes operações: inicia transmissão, inicia difusão, "nohold", permite recepção, verificação das vias, auto diagnóstico.

d) Registro de controle da DMA

Indica o modo de funcionamento da DMA, como por exemplo: byte a byte, rajada, etc.

e) Endereço da memória

Registro da DMA que indica o endereço da memória a ser acessado para uma transferência. Protege contra acesso indevido. Um registro para recepção e outro para transmissão.

f) Contador da DMA

Indica o número de bytes a transferir ou o número de bytes recebidos. Um registro p/ recepção e outro p/ transmissão.

g) Interrupção

O software do processador local deve fornecer uma rotina para tratamento de interrupção vinda do controlador.

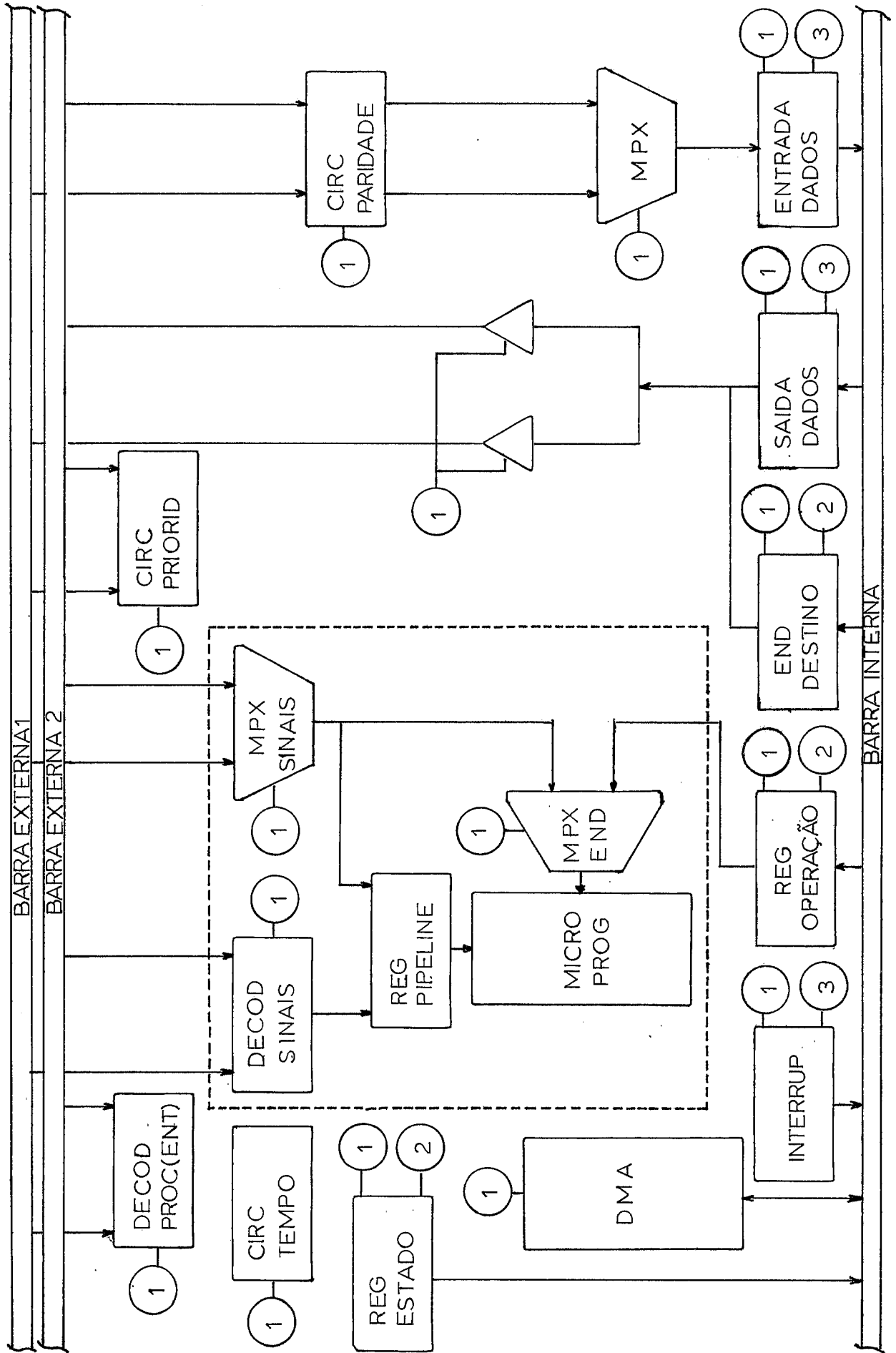


FIGURA (II.10)

1 - SINAIS E/S DO MICROPROC 2 - SINAIS E/S MICROPROC 3 - SINAIS E/S DMA

II.4.4. Comunicação na Arquitetura COMIC

A comunicação entre dois processadores pode ser entendida através dos seguintes procedimentos:

a) A partir da necessidade do envio de um sinal externo o processador local (chamado origem) programa o controlador de comunicação para que o mesmo realize a transferência. Programa a DMA com modo, endereço da memória onde está o sinal e tamanho do sinal. A seguir coloca no registro correspondente o processador destino e finalmente coloca no registro de operações o pedido de transferência. O processador local está liberado.

b) Este pedido inicia a lógica no microprograma para realizar a transferência. No caso de não poder ser realizada a operação, o controle coloca no registro de estado o motivo e aciona a interrupção para avisar o processador origem.

c) Caso não haja impedimento o controlador inicia o ciclo de acesso à barra a fim de ligar o processador origem ao destino. O pedido de acesso é levantado nas duas barras e ao mesmo tempo é iniciado um tempo de "retry". Este tempo de "retry" faz com que o pedido seja retirado e novamente habilitado por intervalos regulares. No caso do pedido ser aceito por uma das barras ele é retirado da outra e o controle assume a barra que o aceitou.

Paralelamente é iniciada uma contagem para saída por tempo ("Hardware") a fim de evitar que a barra e o controle fiquem presos caso o destino não responda. O estouro do prazo gera uma condição de erro e aciona interrupção para o processador origem.

d) Assim que o controlador origem assume a barra ele envia para a mesma o endereço do controlador destino e o pedido de início de transferência.

e) O processador destino deve sempre estar com seu controlador de comunicação programado para poder realizar uma recepção. Cabe mencionar que a recepção é prioritária sobre a transmissão. Existe um canal para recepção e outro para transmissão.

f) Apenas o controlador que corresponde ao endereço jogado na barra terá sua lógica acionada. O endereço juntamente com o pedido de início de transferência faz com que o controlador destino responda que está ligado ao origem, caso possa aceitar a comunicação. Paralelamente é iniciada uma temporização para esperar o primeiro dado. Caso esse dado venha antes de estourar o prazo esse tempo é desligado e outro é acionado a fim de esperar o fim da transferência (tempo máximo de transferência de um sinal). Caso o tempo acabe é acionado erro e interrupção no

processador destino.

g) Chegando a resposta de que o destino esta' ligado, o controlador de origem se prepara para o início da transferência de dados. O microprograma avisa a DMA que vai se iniciar uma transferência de dados e esta pede a barra interna ao processador através de um "hold". O "hold" e' mantido ate' que o último dado seja transferido. Quando o processador fornece a barra interna a DMA coloca o 1o. dado no registro correspondente e avisa o controlador. Este, envia o dado a uma taxa de transferência igual a velocidade da DMA, para o registro de dados de destino. Paralelamente e' iniciado no processador origem uma temporização para esperar o final da transferência de todo o sinal.

h) A chegada do dado e' avisada para o controlador DESTINO, este desliga o tempo de espera do primeiro dado, liga o tempo de espera para transferência do sinal inteiro e avisa a DMA da existência de dado no registro. O aviso da chegada do dado ao registro so' e' indicado se não houver erro de paridade no dado, caso contrário e' assinalado um erro e acionada uma interrupção para o processador destino. A origem também e' avisada do erro.

i) A DMA pede a barra interna ao processador, a qual sera' mantida sob controle da DMA ate' o fim da transferência do sinal, e ao recebe-la adquire o dado do registro, colocando na posição de memória anteriormente programada.

j) Ao ser adquirido o dado, o controle destino avisa o controle origem ("Acknowledge"). No caso de não haver resposta dentro de um tempo programado e' indicado erro e gerada uma interrupção para o processador origem.

l) A transferência continua ate' que o último dado e' transferido ou ate' que ocorra erro.

m) Na transferência do último dado o controlador origem e' avisado pela DMA. Desta maneira ele gera o fim de transferência para o destino para permitir o desligamento.

n) O contador destino ao receber o último dado avisa a DMA normalmente, esta adquire o dado e o coloca na memória. A seguir o controlador gera o "Acknowledge" para a origem, se desliga da origem habilitando nova transferência, na barra desfaz o "hold" da barra interna, desliga a saída por tempo, e coloca o resultado da transferência no registro de estado. Feito isto aciona a interrupção para o processador destino p/ que o mesmo trate o sinal e fica esperando que a atual recepção seja tratada para habilitar outra recepção.

o) O controlador origem ao receber "Acknowledge" desliga o tempo de espera, libera a barra, coloca o resultado da operação no registro de estado e espera nova operação.

p) No caso de ser um difusão os demais processadores atuarão como destino e não haverá "Acknowledge" a cada dado.

II.4.5. Implementação

Para efeitos de implementação será utilizado o microprocessador 8085.

Para DMA será utilizada a pastilha 8237, com quatro canais de DMA independentes.

O registro de processadores será de 6 bits, possibilitando o acesso a até 63 processadores. O processador de número 0 será utilizado para difusão.

Serão implementadas duas barras paralelas.

No caso de difusão haverá apenas a temporização (saída por tempo) para uma transf. total de sinal. O erro de paridade não será avisado à origem e não haverá "Acknowledge" dado a dado.

No caso de sinal comum haverá o tempo de espera dado a dado e o de uma transf. inteira em paralelo.

CAPÍTULO III

CONCORRÊNCIA E LINGUAGENS COM ATRIBUTOS DE CONCORRÊNCIA

III.1. INTRODUÇÃO

Concorrência é a execução de dois processos sobrepostos no tempo, intercalados no tempo ou ambos. Se a primeira operação de um processo começa antes que a última operação de outro processo termine temos dois processos concorrentes. A sobreposição só existe se tiver mais de um recurso do tipo solicitado para executar os processos concorrentes.

Como visto no Cap. I o problema central da concorrência é o compartilhamento dos recursos do sistema (computador). A programação concorrente é uma das maneiras propostas para resolver os problemas de compartilhamento de uma forma eficiente. Essa competição se faz necessária ou pela escassez dos recursos ou pela necessidade dos processos cooperarem entre si. Outra definição de processo concorrente pode ser aquele cuja execução depende de resultados obtidos na execução de outro.

A concorrência é uma propriedade dinâmica, ou seja, ocorre durante a execução de um programa quando dois ou mais processos deste programa são executados ao mesmo tempo. Cabe ressaltar que ao mesmo tempo tem um significado lógico e não necessariamente físico.

As linguagens para concorrência utilizam métodos fixos para estabelecer as unidades concorrentes, tais como processos, tarefas, etc. O processo é portanto a unidade de execução concorrente dentro do qual a execução é sequencial. O processo é uma parte do programa que permite execução concorrente com outros processos do mesmo programa ou de outros programas. Existem basicamente dois tipos de relação entre processos no que diz respeito à execução no tempo. Um tipo de relação é a concorrente, já vista, na qual a execução de um processo depende da execução de outro processo. O outro tipo é a relação disjunta, cuja execução paralela de processos tem o mesmo resultado da execução sequencial. O primeiro tipo é o objeto de nosso interesse.

Para permitir a concorrência é necessária a existência de um módulo que controle os recursos compartilhados alocando-os aos processos quando possível, de maneira organizada a fim de garantir

a confiabilidade em termos de segurança e rapidez. Este módulo, que chamaremos de núcleo, deve implementar as funções de controle dos recursos compartilhados, sincronismo e comunicação entre processos de um mesmo programa, de programas diferentes ou de processadores diferentes no caso de sistema distribuído. Várias são as maneiras já propostas para implementar as funções acima entre as quais podem ser citadas as mensagens, os sinais, os semáforos, região crítica, etc. O núcleo é o módulo que implementa os meios descritos, sendo em última instância, o responsável pela eficiência, confiabilidade e capacidade do sistema.

A programação concorrente é portanto uma importante ferramenta para melhorar a capacidade dos sistemas.

Vários são os tipos de sistemas que podem se beneficiar, ou mesmo que se tornaram possíveis de implementar através da utilização do conceito de concorrência e sistema distribuído. O sistema para controle de processos em tempo real é um exemplo de um tipo e a telefonia (CPA-T) uma aplicação.

III.2. LINGUAGENS CONCORRENTES

Várias têm sido as linguagens propostas com mecanismos para o controle e implementação dos objetivos de concorrência anteriormente descritos. Verifica-se porém que as várias linguagens foram implementadas na sua maioria em máquinas com um único processador não se ocupando especificamente da existência de sistemas distribuídos ou mesmo de co-processamento (E/S, aritmético).

As linguagens que implementam aspectos de concorrência têm em geral tres conceitos:

a) Sincronização

Sincronização é a coordenação de processos, ou seja, um processo pode ser retardado até que outro atinja um determinado ponto para então reativar o anterior.

b) Compartilhamento de recursos

O compartilhamento de recursos é necessário devido a escassez dos mesmos e da necessidade de fornecer aos processos concorrentes igualdade na sua utilização. Este conceito, que em muitos casos exige exclusão mútua, pode ser implementado através de monitores, regiões críticas, semáforos como já proposto em algumas linguagens. A região

crítica e' um trecho de um processo que faz acesso à um recurso compartilhado. Para garantir que apenas um processo acessa o recurso de cada vez e' empregado o conceito de semáforo que checa a execução de uma determinada região crítica por somente um processo.

c) Comunicação

Este aspecto tem como interesse a troca de informações entre processos. Estes processos podem pertencer a um mesmo programa, a programas diferentes ou a processadores diferentes. Existe dois tipos de comunicação: síncrona e assíncrona. Na síncrona o processo ao enviar uma informação para outro fica bloqueado em uma determinada condição esperando a recepção. Na assíncrona o processo que envia a informação continua a sua execução sem parar após a comunicação.

Segue-se uma descrição breve do tratamento dado à concorrência por algumas linguagens conhecidas, tais como PASCAL, MODULA, ADA, CHILL.

III.2.1. Pascal Concorrente

O Pascal Concorrente e' uma linguagem proposta por BRINCH HANSEN que implementa aspectos de concorrência. A sua utilização e' acadêmica, servindo de base para estudos e pesquisas no assunto. Foi uma das primeiras propostas.

Uma preocupação evidente no Pascal Concorrente e que levou à sua forma e' a eliminação da maior parte possível dos erros de programação dependente do tempo durante a compilação. Se por um lado isto leva à maior segurança, leva também à uma rigidez na linguagem, já que esse objetivo e' realizado através da rígida modularização dos programas em Pascal Concorrente. A modularidade dos programas em Pascal Concorrente e' suportada pelos conceitos de monitor, processo e classe.

Outro aspecto que surge no Pascal Concorrente e' a sua adequação para utilização em sistemas com um único processador e com memória compartilhada. Este aspecto vem do conceito de monitor utilizado pelo Pascal Concorrente para troca de informações entre unidades executivas.

A seguir uma descrição do tratamento dado pelo Pascal Concorrente aos aspectos de concorrência.

a) Ativação e desativação de uma unidade executiva.

No Pascal Concorrente a unidade executiva é chamada Processo. O processo é um programa sequencial que somente tem acesso a suas variáveis e parâmetros. A ativação de um processo é realizada por meio de uma instrução INIT "process". Um processo só pode ser iniciado uma vez e, ao ser iniciado, suas variáveis privadas e parâmetros são criados para sempre (variáveis permanentes). A instrução INIT pode dar início à execução de mais de um processo.

```
INIT   "Process 1"  (      )
      "Process 2"  (      )
      "Process 3"  (      )
```

As variáveis e parâmetros do processo são estabelecidos em tempo de compilação mediante declarações.

Um processo em Pascal Concorrente não é desativado. Uma vez iniciado o processo existe para sempre.

b) Sincronização de Unidades Executivas

O Pascal Concorrente dispõe para a sincronização de unidades executivas das instruções DELAY, CONTINUE realizadas sobre QUEUES (filas) e do conceito de monitor. O monitor é um conjunto de rotinas cuja responsabilidade é gerenciar um recurso. Quando um processo deseja utilizar um recurso, ele faz uma chamada ao monitor correspondente. Se vários processos desejam executar procedimentos do monitor simultaneamente, estes procedimentos serão executados um de cada vez, em uma ordem não especificada. Desta forma o monitor impõe a exclusão mútua na sincronização de operações realizadas por processos concorrentes.

A instrução DELAY (Q) suspende a execução do atual processo na espera de uma determinada condição. O processo é colocado em uma fila (QVEUE) relativa a essa condição. A instrução CONTINUE (Q) reativa o processo que estava suspenso na espera da condição.

Com as instruções e as condições acima, o monitor realiza a sincronização e comunicação dos processos.

c) Comunicação entre Unidades Executivas

E' realizada através do compartilhamento de dados declarados internos a um monitor. Desta maneira dois processos podem trocar informações através de um buffer controlado por um monitor. A parte do processo que trabalha com os dados compartilhados (buffer) e' conhecida como região crítica e deve ser executada somente com permissão do monitor correspondente.

d) Exclusão Mútua

A exclusão mútua sobre dados compartilhados e' garantida no Pascal Concorrente pelo Monitor.

III.2.2. Modula

Modula e' uma linguagem de programação preliminarmente projetada para a implementação de sistemas. Idealizada por N. Wirth para implementação em sistemas com um único processador, a linguagem Modula possui uma versão antiga chamada Modula e uma versão mais nova chamada Modula-2. A Modula-2 herda da versão anterior o importante conceito de módulos e baseia a sua estrutura de dados na linguagem PASCAL.

A diferença básica entre as duas versões esta' no tratamento da comunicação e sincronização entre processos. A primeira versão introduz o conceito de módulos interfaces e condições (SIGNAL) manipuladas por operações de WAIT e SEND para realizar comunicação e sincronização entre processos. Esse conceito e' parecido com o conceito de monitor utilizado em Pascal Concorrente.

A segunda versão fornece facilidades básicas para a implementação da comunicação e sincronismo entre processos através do conceito de co-rotinas. A vantagem e' permitir ao projetista selecionar o algoritmo de escalonamento de processos que mais convier às suas necessidades.

A linguagem Modula (Modula 2) e' independente da máquina. Ela foi projetada com o intuito de permitir escrever sistemas em todos os níveis, eliminando totalmente a necessidade de programar em LINGUAGEM DE MAQUINA, mesmo em níveis mais baixos. Para resolver o problema das operações dependentes da máquina, estas são introduzidas em módulos específicos e desta maneira são isoladas. Tais operações (E/S, escalonamento de processos, INTERRUPTÃO, etc) constituem uma coleção de módulos padrão para a máquina na implementação do sistema em Modula.

A linguagem Modula também apresenta facilidades para a programação de grandes sistemas. O módulo e' dividido em duas

partes. Uma parte é a especificação do módulo que contém a interface do módulo com o exterior. A outra parte é a implementação do módulo.

A seguir são apresentados os aspectos da linguagem Modula relacionados com concorrência.

a) Ativação e desativação de Unidades Executivas

A unidade executiva na linguagem Modula (Modula 2) é o processo. O processo é uma procedure de um módulo. Em Modula, 2 processos são co-rotinas que compartilham um único processador. Nas duas versões um novo processo é criado por uma instrução que especifica o nome do processo.

NEWPROCESS (A; B; C; D)

onde A é o procedimento que compõe o processo

B é o endereço base do processo

C é o tamanho da área de trabalho

D é o rótulo ligado ao procedimento para identificá-lo como processo.

A ativação só pode ser feita no corpo do programa principal não havendo aninhamento. Pode haver transferência de controle de um processo para outro.

TRANSFER (P1; P2)

Um processo termina quando é executada a instrução END do seu corpo.

b) Sincronização de unidades executivas

Existe uma diferença entre as duas versões quanto a esse aspecto. Em Modula 2 o modo de implementação da sincronização entre processos fica a cargo do projetista a partir de ferramentas básicas fornecidas pela linguagem. Em Modula o sincronismo é implementado por INTERFACES MODULES que realizam operações SEND e WAIT sobre condições SIGNAL. Este conceito é análogo ao conceito de monitores do PASCAL. O sincronismo entre processos é garantido pela execução destas operações sobre as condições dentro

dos módulos interfaces. É possível também executar uma operação WAIT(s) especificando prioridade. A operação AWAIT(s) restitui "TRUE" se existir pelo menos um processo em espera pela condição fornecida. Essas duas últimas ferramentas fornecem um não determinismo na implementação do sistema.

c) Comunicação entre unidades executivas.

Existe uma diferença entre as duas versões quanto a esse aspecto. Em Modula 2 cabe ao projetista o modo de implementação da comunicação entre processos. Em Modula a comunicação é realizada por meio de dados compartilhados declarados internamente aos módulos interfaces. As operações SEND e WAIT e as condições SIGNAL fornecem o sincronismo necessário para a comunicação (análogo ao conceito de monitores do PASCAL).

d) Exclusão Mútua

Também nesse aspecto existe a mesma diferença entre as duas versões. Modula se utiliza do conceito de interface Module, das operações SEND e WAIT e das condições SIGNAL para implementar exclusão mútua. Análogo ao conceito de monitor do PASCAL.

As operações de E/S são implementadas através do conceito de device module permitindo ao programador escrever procedimentos para periféricos. A device module engloba em seu corpo o device process e o(s) device(s) register(s). Tanto o device quanto o(s) device(s) register(s) só podem ser declarados dentro do corpo de um único device module, e só no device module, garantindo dessa maneira a integridade. A comunicação do device process com outro processo qualquer é feita via um procedimento declarada no device module e disponível externamente.

III.2.3. ADA

A linguagem ADA foi desenvolvida pela empresa Cii Honeywell Bull para o Departamento de Defesa dos EUA. O objetivo da linguagem é o de fornecer uma ferramenta padrão para o desenvolvimento de software para aplicação em sistemas embutidos (físicamente incorporado em um sistema mais amplo cuja função principal não é processamento de dados e que não pode ser projetado, implementado e operado sem considerar seus componentes computacionais) que possibilite a padronização, a redução dos custos de programação e facilite a implementação (modularidade). A necessidade da existência de uma linguagem como a ADA surgiu dos grandes gastos realizados no desenvolvimento do software para os sistemas mencionados e da diversidade de linguagens encontradas nos diferentes produtos tornando difíceis a manutenção, alteração e portabilidade.

Por pretender ser uma linguagem padrão em seu campo de aplicação, a linguagem ADA implementa aspectos comuns à várias linguagens, além de outros não existentes. Isto a torna de certa forma uma linguagem complexa. Para facilitar o desenvolvimento da ADA foi escolhida a linguagem PASCAL como base.

A seguir daremos uma descrição dos aspectos da ADA ligados à concorrência.

a) Ativação e desativação de unidades executivas

A concorrência na linguagem ADA é suportada por um conceito chamado TASK. Uma TASK é constituída de duas partes. A primeira parte é a especificação da TASK, onde é descrita a interface da TASK com o exterior. A segunda parte é o corpo da TASK, o qual contém a sequência de instruções que descreve o comportamento interno da TASK. O cabeçalho com o nome da TASK, a declaração de constantes, tipos, subprogramas, exceções e entradas (ENTRY) associadas à TASK e visíveis externamente estão na especificação da TASK. Por sua vez no corpo da TASK temos a declaração das estruturas de dados locais e as sequências de comandos que implementam as entradas descritas na especificação.

Ex:

```
TASK MENSAGEM IS
  TAMANHO: CONSTANT INTEGER: = 256;
  TYPE PACOTE IS ARRAY (1...TAMANHO) OF CHARACTER;
  ENTRY READ (V: OUT PACOTE);
  ENTRY WRITE (E: IN PACOTE),
```

```
END MENSAGEM
```

```
TASK BODY MENSAGEM IS
  TAMENSAGEM: CONSTANT INTEGER: = 10;
  MENS      : (1...TAMENSAGEM) OF PACOTE;
  IN, OUT: INTEGER. RANGE. 1... TAMENSAGEM: = 1;
  CONTADOR: INTEGER. RANGE. 1...TAMENSAGEM: = 0;
```

```
BEGIN
  COMANDOS QUE IMPLEMENTAM ENTRIES READ E WRITE
END MENSAGEM;
```

Na linguagem ADA podemos ter uma hierarquia entre as TASKS. A TASK que em seu corpo englobe declarações de outras TASKS é chamada PAI destas TASKS. Da mesma forma, as TASKS assim

declaradas são descendentes do PAI. Uma TASK e' ativada a partir de um comando INITIATE que pode ser dado no corpo de outra TASK.

```
INITIATE MENSAGEM;
```

Uma TASK pode ser terminada de dois modos. Pode terminar em um comando TERMINATE, sendo que se a TASK for PAI de outras TASKS e as descendentes não tiverem terminado a TASK PAI sera' atrasada. Pode também terminar por um comando ABORT TASK. Neste caso a TASK e suas descendentes, se houver, serão terminadas incondicionalmente. Uma nova proposta para a linguagem ADA elimina os comandos INITIATE e ABORT. Nesta proposta a TASK PAI e suas TASKS descendentes são automaticamente ativadas quando a TASK pai atinge o BEGIN de seu corpo. As TASKS são terminadas normalmente em um END. Uma tarefa pai so' esta' terminada quando ela e suas descendentes executarem os END's em seus corpos.

A linguagem ADA permite a criação de uma familia de TASKS. Esta familia e' um conjunto de TASKS iguais diferenciadas apenas por um rótulo na chamada. Um exemplo típico da necessidade desse conceito e' quando temos vários periféricos iguais cada um controlado por uma TASK distinta.

```
TASK TELETYPE (1 ... 20) is
END TELETYPE;
```

```
INITIATE TELETYPE (5);
```

Uma tarefa também pode ser declarada como tipo. Após isto podem ser ativadas as duas encarnações mencionando-se o tipo na ativação.

Outra facilidade introduzida na linguagem ADA e' a possibilidade de ler e alterar a prioridade de uma TASK conforme a necessidade, durante a execução desta TASK o que fornece um não determinismo.

b) Sincronização de unidades executivas

A sincronização na linguagem ADA e' realizada através do conceito de rendez-vous (encontro) entre duas TASKS. Uma tarefa requer o encontro de uma outra mediante uma instrução CALL referente a uma ENTRY da outra. A outra aceita o encontro através

de execução do comando ACCEPT referente a ENTRY chamada. A chamada de uma ENTRY e' similar a chamada de uma procedure.

A TASK que executa a chamada da ENTRY fica suspensa ate' que a TASK destino execute o ACCEPT referente a entrada chamada. Da mesma forma a TASK que executa um ACCEPT relativo a uma entrada não chamada fica suspensa ate' que a chamada ocorra. Para fornecer maior flexibilidade uma TASK pode esperar por uma chamada em um número alternativo de ENTRIES através do comando SELECT. O comando SELECT associado a clausula WHEN permite testes de condições lógicas para que a TASK possa então executar diferentes procedimentos conforme o caso. Condições lógicas são um importante conceito dentro de um esquema de sincronização.

```

- TASK BODY MENSAGEM IS
    -
    -
    -
- BEGIN
  LOOP
    SELECT
      WHEN CONTADOR  TAMANHO
        ACCEPT WRITE (E: IN PACOTE) DO
          BUF (INX): = E;
        END WRITE;
        INX: = INX MOD TAMANHO + 1;
        CONTADOR: = CONTADOR + 1;
      OR WHEN CONTADOR  0
        ACCEPT READ (V: OUT PACOTE) DO
          v: = BUF (OUT);
        END READ;
        OUT: = OUT MOD TAMANHO + 1;
        CONTADOR: = CONTADOR - 1;
    END LOOP;

```

.END MENSAGEM;

Outro comando existente na linguagem ADA para realizar sincronização é o DELAY. Este comando permite atrasar uma TASK por um período de tempo pedido.

c) Comunicação entre unidades executivas

O conceito de rendez-vous descrito anteriormente representa também um instrumento de comunicação entre TASKS porquanto, uma TASK que executa um CALL ENTRY pode passar parâmetros de informação à TASK que executa o ACCEPT ou SELECT. Além disso a possibilidade de passar parâmetros por referência permite à TASK que executa o ACCEPT restituir informações à TASK que a chamou. Desta forma a comunicação entre TASK é possibilitada por ENTRY CALL, SELECT e ACCEPTS.

Duas possibilidades podem ocorrer em uma comunicação na linguagem ADA devido ao conceito de rendez-vous. A TASK que chama pode realizar a ENTRY CALL antes ou depois do correspondente ACCEPT ser atingido pela TASK chamada. Em qualquer dos casos a TASK que atingir o rendez-vous (encontro) é retardada até que a outra também o atinga. Quando o rendez-vous é atingido os parâmetros de entrada são passados para a TASK chamada e a TASK que chama é suspensa enquanto a TASK chamada executa o corpo do ACCEPT. Após a execução do corpo do ACCEPT parâmetros de saída podem ser retornados, a TASK que chamou, e as duas TASKS prosseguem independentemente.

Uma fila de TASKS por ENTRY pode ser criada para lidar com o caso de mais de uma TASK realizando a mesma ENTRY CALL e ainda não atendidas. Uma TASK só pode estar associada à uma fila uma vez que só pode executar uma ENTRY CALL de cada vez. Quando ocorre o ACCEPT referente a ENTRY a TASK mais prioritária é retirada da fila.

Duas conclusões são retiradas da descrição acima. A primeira é que a comunicação na linguagem ADA é síncrona e assimétrica, já que a TASK chamada não precisa saber o nome da TASK que a chamou. A segunda é que sendo o corpo do ACCEPT executado por exclusão mútua, este deve ser o menor possível.

d) Exclusão mútua

O conceito de rendez-vous é suficientemente abrangente para

possibilitar a realização de exclusão mútua sobre objetos compartilhados através da criação de uma TASK que gerencie esses dados.

III.2.4. CHILL

A linguagem CHILL é uma proposta do CCITT, órgão internacional de telecomunicações, para uma linguagem padrão de alto nível e com atributos de concorrência para uso em telecomunicações e que venha facilitar a programação, implementação e manutenção dos cada vez mais complexos sistemas do campo de interesse acima citado.

A linguagem CHILL foi idealizada com a intenção de fornecer uma importante ferramenta para a programação de sistemas SPC (CPA-T), nos quais os tamanhos de programas, grau de confiabilidade e elevado grau de paralelismo são requisitos importantes.

Em vista disso, o CHILL possui uma mistura de diferentes mecanismos para controle de concorrência tais como, comunicação entre unidades executivas, acesso disciplinado a recursos compartilhados e sincronização, visando a aplicação em sistemas distribuídos.

Por ser uma linguagem que pretende implementar várias proposições feitas em outras linguagens, o CHILL torna-se uma linguagem para o projetista de software. Isto é devido mais a gama variada de aplicações a ser atingida do que a complexidade da linguagem. Isto pode ser resolvido através da escolha de um sub-conjunto adequado para a aplicação específica.

A linguagem CHILL embora projetada para sistemas de telecomunicações pode ser aplicada em outros campos devido a sua flexibilidade e poder. Pode ser usada para projetar sistemas operacionais, sistemas de aplicação, suporte, teste e manutenção, etc.

Alguns aspectos importantes da linguagem CHILL podem ser citados: Ela é independente da máquina, permite modularização (programação estruturada), tem alta confiabilidade (verificação dos aspectos de concorrência durante a compilação) e possui geração de código eficiente. Todos esses aspectos são levados em conta no CHILL devido ao software de sistemas SPC estarem entre os maiores e mais complexos e com tendência a aumentar em tamanho e complexidade no futuro. Aliado a esses fatos está a necessidade de alta confiabilidade nesse tipo de aplicação.

Outro aspecto da linguagem CHILL e' a padronização que ela fornece para sistemas de telecomunicações. Essa padronização e' importante, ja' que atualmente existem vários sistemas com diferentes linguagens, que devido à universalidade das telecomunicações tem que interagir e coexistir provocando dificuldades para a interligação, manutenção e alteração dos mesmos.

Um programa CHILL e' constituído por um módulo. Cada módulo pode ser formado por processos, procedures, regiões críticas e dados. Um processo, por sua vez, e' constituído de procedures, dados e ações e a região crítica e' formada por procedures e dados. Dois importantes aspectos são introduzidos na linguagem CHILL. O primeiro e' a visibilidade de um nome, que significa os pontos do programa onde um nome pode ser usado. O segundo e' o tempo de vida de uma posição, que significa o tempo de existência de uma dada posição para o programa. O módulo fornece meios para implementar a visibilidade de um nome contra uso indevido. A procedure e' um sub-programa que pode ser chamada de qualquer ponto do programa. A região crítica e processos fornecem meios para a programação concorrente.

A seguir veremos alguns aspectos da linguagem CHILL no que diz respeito a concorrência.

a) Ativação e Desativação de unidades executivas

A concorrência em CHILL e' suportada por unidades executivas chamadas processos. Um programa CHILL consiste de um ou mais processos, sendo que dentro de um processo a execução e' sempre sequencial. Um processo e' visto pela linguagem CHILL segundo dois aspectos. O primeiro aspecto e' o estático, o qual recebe o nome de definição do processo. Na definição do processo estão descritas as ações e objetos locais do processo, sendo a partir da definição que o mesmo e' criado.

```
Ex:  ALARME LOCAL:
      PROCESS (      );
      -
      -
      END ALARME LOCAL;
```

Um programa CHILL e' constituído de um processo externo e processos internos. O processo externo e' definido como tudo que não se encontra dentro dos processos internos.


```

Ex: C          β
    A          β
    PROCESS(  );β INTERNO
    -          β
    END A;     β          PROGRAMA

    B          β
    PROCESS(  );β
    -          β INTERNO
    END B     β

EXTERNO β -
        β -
        β END C;

```

Não pode haver definição de outros processos dentro dos processos internos.

O segundo aspecto é o dinâmico e se refere a introdução do conceito de instância. A instância é a encarnação do processo, ou seja, é a versão executável do processo. Desta forma podemos associar uma variável, declarada como instância, a uma determinada ativação de um processo, tendo diferentes versões executáveis de uma mesma definição de processo. Isto pode ser entendido com o seguinte exemplo:

Um processo trabalha sobre uma base de dados a partir do pedido de dois possíveis terminais de vídeo e responde ao que perguntou. Os terminais de vídeo podem fazer perguntas a qualquer tempo. Pode-se notar que é interessante ter apenas um processo, com a possibilidade de associar em tempo de execução o terminal que fez a pergunta a fim de fornecer a resposta ao terminal correto.

Ex: declaração de instância

```
DCL LOC INSTANCE
```

Para se obter a identidade do processo em execução usa-se o seguinte operador:

```

A
PROCESS ( );
-
LOC: = THIS;
-
END A;

```

Este comando associa a variável LOC (INSTANCIA) ao processo em execução. A instância também pode ser associada na ativação do processo.

```
LOC: = START A;
```

Um processo em CHILL deve estar em um dos três seguintes estados: desativado, ativo ou atrasado. As possíveis transições, causadas por ações no programa, podem ser vistas na Figura (III.1).

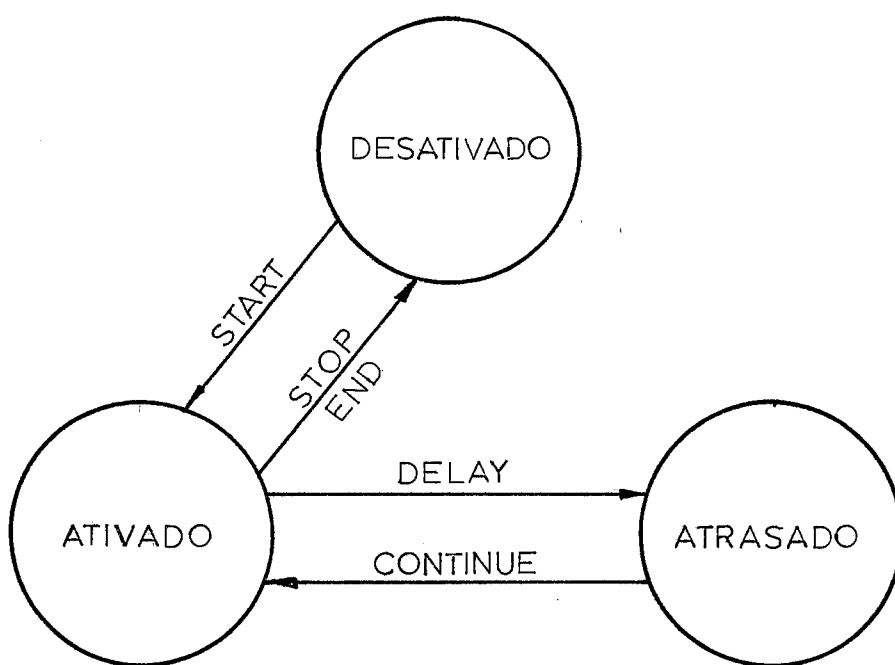


FIGURA (III.1)

O comando START ativa um processo a partir da definição do mesmo. Esse processo é executado concorrentemente com o processo que o ativou.

A ativação do processo externo de um programa CHILL é realizada automaticamente pelo sistema. Os processos internos podem ser ativados pelo processo externo ou por outros processos internos.

```
START A ( );
```

Um processo pode terminar (desativado) de duas maneiras. Executando a instrução END ao final do seu corpo ou através do comando STOP por ele executado. Um processo não pode desativar outro.

A execução da instrução DELAY sobre eventos, a recepção ou envio de ações sobre "buffers" ou a recepção de sinais podem causar o atraso do processo executante. Da mesma forma, a execução da instrução CONTINUE sobre eventos, o envio de ações sobre buffers ou sinais podem levar o processo de atrasado para ativado.

b) Sincronização de unidades executivas

A sincronização de processos na linguagem CHILL pode ser atingida através de eventos em combinação com instruções DELAY e CONTINUE realizadas sobre esses eventos. Eventos são condições declaradas em um programa.

DCL OPERATOREADY EVENT

Pode existir uma fila de processos esperando por um evento. Na ocorrência do evento um processo da fila, provavelmente o primeiro, é retirado da fila e ativado. A sincronização ocorre desta maneira: um processo se suspende na espera de um evento ou sinaliza que um evento ocorreu a fim de ativar um processo que esteja esperando pelo evento.

Um processo pode esperar por vários eventos através do comando DELAY CASE, fornecendo um não determinismo na programação.

Podem ser associadas prioridades ao comando DELAY a fim de prover maior flexibilidade. Se a prioridade 1 é especificada, o processo suspenso é colocado no topo da fila relativa ao evento.

Se não ha' prioridade especificada, ou a mesma e' 0, o processo e' colocado no fim da fila (menos prioritário).

A instrução DELAY e' sempre bloqueante pois o processo que a executa se suspende. A instrução CONTINUE e' não bloqueante e não persistente, ou seja, o processo continua executando e qualquer DELAY relativo ao evento que ocorra após a sinalização do mesmo coloca o processo à espera da nova ocorrência do evento (no caso de não haver processo na espera nada acontece).

Sinais e "buffers" podem ser usados para sincronização, porém eventos não podem ser usados para comunicação.

c) Comunicação entre unidades executivas

A comunicação entre processos na linguagem CHILL e' realizada através de dois conceitos: Buffer e Sinal. Buffers são alocados pelo usuário no seu programa e mensagens podem ser enviadas ou recebidas via buffers. Buffers podem ser declarados da seguinte maneira:

```
DCL CAIXA BUFFER (10);
```

O usuário pode alocar vários buffers de um mesmo tipo. Embora a alocação dos buffers seja feita pelo usuário, o controle dos mesmos e' feito pelo sistema automaticamente, ou seja, e' o sistema que sabe se um buffer esta' cheio ou vazio, e que processos estão à espera de determinado buffer.

Três operações são realizadas sobre os buffers:

SEND, RECEIVE e RECEIVE CASE

A operação SEND pede o envio de um buffer. Se o buffer esta' cheio o processo emissor e' atrasado ate' que o buffer esteja disponível.

```
SEND CAIXA;
```

A operação RECEIVE pede a recepção de um buffer. Se o buffer

esta' vazio o processo receptor e' atrasado ate' que o buffer esteja cheio.

```
RECEIVE CAIXA;
```

A operação RECEIVE CASE permite não determinismo ao oferecer a possibilidade de um processo esperar por mais de um buffer.

Buffers ao contrário de eventos são persistentes, isto e', se uma mensagem e' enviada e não ha' processo à espera para recebe-la, ela e' armazenada no buffer ate' que seja aceita. O processo que envia não e' atrasado. Uma prioridade pode ser ligada a um buffer para prover maior flexibilidade. Um buffer sem mensagem pode ser usado para sincronização.

Sinais são usados para comunicação e sincronização de processos. Sinais são alocados e controlados automaticamente pelo sistema. São definidos por um comando da seguinte forma:

```
SIGNAL S = (INT, BOOL);
```

Tres operações são realizadas sobre um sinal:

```
SEND, RECEIVE e RECEIVE CASE
```

A operação SEND pede o envio de um sinal, sendo que a alocação e' automática (sempre deve haver sinal disponível) e o processo destino deve ser especificado.

```
SEND S (5) TO PI;
```

O processo emissor nunca e' atrasado.

A operação RECEIVE pede a recepção de um sinal e pode provocar a suspensão do processo em uma fila a espera do sinal.

A operação RECEIVE CASE fornece a possibilidade de um processo esperar por mais de um sinal.

Sinais são persistentes e podem ter prioridades associadas. Um sinal sem mensagem pode ser usado para sincronização.

São 3 as diferenças principais entre buffer e sinal:

- Sinais são enviados para um destino específico, buffers não
- Sinais são alocados automaticamente pelo sistema, enquanto buffers são alocados pelo usuário
- Um processo ao enviar um sinal nunca é bloqueado, porém ao enviar um buffer ele pode ser suspenso.

Estes conceitos são de uma certa maneira similares e dependendo da aplicação somente um deles pode ser suficiente.

d) Exclusão mútua

A exclusão mútua entre processos na linguagem CHILL pode ser implementada através de uma construção especial chamada REGION. Os objetos a serem guardados contra acesso concorrente são declarados internos à região e somente podem ser acessados por procedures críticas, isto é, definidas dentro da região e chamadas por outros processos. Se um processo tenta acessar um objeto compartilhado, por meio de uma procedure crítica chamada anteriormente por outro processo ele é retardado até que a região esteja livre. Desta maneira garante-se que somente um processo tem acesso a região de cada vez. Uma região não pode ser declarada dentro de outra região.

ALOCACAO DE RECURSOS:

REGION

DCL RECURSO LIVRE EVENT;

-
 ALOCATE :
 PROC () (INT);

-
 END ALOCATE;
 DE ALOCATE:
 PROC () ;

-
 END DE ALOCATE;

END ALOCACAO DE RECURSOS;

III.2.5. Conclusão

Existem outras proposições de linguagens para tratamento dos aspectos de concorrência, porém as linguagens anteriormente apresentadas são suficientes pois abrangem áreas de aplicação diversas, fornecem uma síntese das ferramentas mais utilizadas para a implementação de concorrência, além de serem bem conhecidas.

Como exemplos de outras linguagens com aspectos de concorrência podemos citar ainda:

MESA, EDISON, PEARL, DISTRIBUTED PROCESS, MOD E ILIAD

III.3. Resumo das Linguagens Apresentadas Quanto aos seus Aspectos de Concorrência.

Segue-se nesta parte um resumo para efeitos de comparação entre as linguagens.

a) PASCAL

- a.1) Ativação/desativação - INIT / não desativa
- a.2) Sincronização - DELAY, CONTINUE SOBRE QUEDES e MONITORES
- a.3) Comunicação - MONITORES
- a.4) Exclusão mútua - MONITORES

b) MODULA

- b.1) Ativação/desativação - NEW PROCESS / END
- b.2) Sincronização - SEND, WAIT SOBRE QUEDES e INTERFACE MODULES
- b.3) Comunicação - INTERFACE MODULES.
- b.4) Exclusão mútua - INTERFACE MODULES

c) ADA

- c.1) Ativação / desativação - INITIATE / ABORT, END

- c.2) Sincronização - REDEZ VOUS, ENTRY, ACCEPT, SELECT e WHEN
- c.3) Comunicação - RENDEZ VOUS, ENTRY, ACCEPT, SELECT e WHEN
- c.4) Exclusão mútua - RENDEZ VOUS, ENTRY, ACCEPT, SELECT e WHEN
- d) CHILL
 - d.1) Ativação / desativação - START / STOP, END
 - d.2) Sincronização - DELAY, CONTINUE, DELAY CASE SOBRE EVENTS (BUFFERS e SINAIS)
 - d.3) Comunicação - SEND, RECEIVE, RECEIVE CASE SOBRE BUFFERS OU SINAIS
 - d.4) Exclusão mútua - REGION

CAPÍTULO IVLINGUAGEM BASE E ATRIBUTOS DE CONCORRÊNCIA IMPLEMENTADOSIV.1. AVALIAÇÃO DAS LINGUAGENS APRESENTADAS QUANTO AOS ASPECTOS DE CONCORRÊNCIA

Uma avaliação da eficiência das linguagens, no que diz respeito à concorrência, deve atentar para a aplicação a ser dada ao sistema proposto. Essa avaliação apesar de restringida pela aplicação é de difícil concepção. Deve ficar claro neste ponto que o sistema aqui proposto tem a sua aplicação voltada para tempo real baseado em multiprocessamento (arquitetura de micros), com interesse maior na área de telecomunicações. Este fato torna necessário recolocar algumas definições já apresentadas e apresentar outras antes de passar à avaliação.

Características intrínsecas à um sistema de multiprocessamento:

- São capazes de aumentar a capacidade efetiva de trabalho porque eles admitem execução independente de tarefas em paralelo.
- Aumentam a complexidade da programação, tornando porém certas aplicações viáveis.
- Deve ser projetado para incluir mecanismos de confiabilidade que se aproveitam da redundância inerente ao "hardware", reconfigurando o sistema conforme a ocorrência de falhas.
- Deve ser modular, permitindo a ampliação conforme exigência da aplicação.

Sistemas de tempo real são aqueles aplicados às condições reais. Um sistema de tempo real recebe entradas de maneira aleatória devendo fornecer respostas dentro de um intervalo de tempo específico a partir de requisitos de confiabilidade rígidos.

Algumas características intrínsecas dos sistemas de tempo real:

- Interage com um meio no qual fatos acontecem simultaneamente e em alta velocidade

- Deve responder de uma maneira não determinística 'as entradas, não especificando de maneira rígida a ordem de atendimento
- Executa tarefas em paralelo
- Deve funcionar sempre e com alta confiabilidade pois paradas e erros podem ser catastróficos.

Características de linguagens em alto nível com aspectos de concorrência. Algumas características são especialmente importantes dentro da aplicação em sistemas de multiprocessamento para tempo real.

- Deve minimizar o tempo de execução, principalmente do núcleo e a geração de código
- Deve oferecer o maior número possível de opções no que diz respeito ao controle de concorrência (ativação, comunicação, sincronização) permitindo 'a aplicação a escolha do sub-conjunto adequado a ela
- Deve ser modular permitindo programação estruturada e projeto em partes evitando erros e viabilizando grandes projetos
- Permitir não determinismo em suas opções e flexibilidade nos algoritmos de controle (escalonamento de tarefas)
- Prover alta confiabilidade e compilação separada evitando o maior número possível de erros na compilação e viabilizando grandes projetos
- Ser independente da máquina o quanto possível
- Permitir a programação em todos os níveis através, por exemplo, da construção de operações para o tratamento de E/S, interrupções, exceções, etc.
- Implementar recursos variados para a criação, ativação e desativação de tarefas paralelas aumentando a flexibilidade do sistema, atingindo uma variedade de aplicações
- Implementar operações simples para comunicação e sincronismo entre processadores com memória distribuída (devido 'a aplicação)
- Ser uma linguagem o mais geral possível, permitindo a sua utilização em campos variados (projeto de sistemas operacionais, suporte, aplicações em tempo real, tempo compartilhado, etc.)

Temos agora uma base para a avaliação da eficiência de uma linguagem, no que diz respeito 'a concorrência, para aplicação em sistemas distribuídos de tempo real. E' necessário enfatizar que

essa avaliação não é exaustiva nem definitiva.

a) Pascal Concorrente

Algumas restrições podem ser feitas ao Pascal Concorrente, entre as quais podemos citar:

- Fornece apenas um meio de se comunicar processos que é o monitor. O monitor necessita de memória compartilhada para comunicação e embora seja um meio seguro de comunicação se adapta melhor à aplicação em sistemas com memória compartilhada.
- O conceito de monitor leva à uma certa inflexibilidade quanto à escalonamento e provoca restrições ao seu uso (política de escalonamento está embutida no núcleo da linguagem).
- As operações de E/S como criadas provocam a necessidade de se alterar o núcleo sempre que um periférico é incluído ou alterado no sistema (o "driver" é implementado no núcleo)
- Não oferece operações de manuseio de exceções, ou seja, no caso de erros ("over-flow", endereçamento, etc.) não existe procedimento embutido na linguagem.

O Pascal apresenta algumas vantagens, tais como:

- A modularidade, que é suportada por conceitos rígidos como o monitores, processos e classes
- Estes conceitos de modularidade e o fato da comunicação só ser feita via monitor possibilitam a verificação em tempo de compilação de muitos erros difíceis de encontrar na execução. Isto leva à confiabilidade e segurança na programação
- A operação de E/S é implementada no núcleo deixando a programação em alto nível alheia a esse fato. A chamada de uma operação de E/S provoca a suspensão do processo que chamou até que a operação seja completada. A operação de E/S é uma operação sequencial do processo como uma outra qualquer (apesar de colocada aqui como vantagem, pode em certas aplicações, tais como projeto de sistemas ou sistemas que exigem mudanças constantes nos periféricos, ser uma desvantagem).

b) Modula

Uma avaliação da linguagem quanto a alguns aspectos leva a algumas restrições:

- É dirigida para programação de sistemas (multiprogramação) com um único processador, principalmente para sistemas que não

necessitem de memória virtual, manuseio dinâmico de processos, variação no tempo de execução e manuseio de prioridades dos processos

- Inflexibilidade quanto 'a criação, destruição de processos. Um processo so' pode ser ativado no corpo do programa principal e a desativação so' ocorre com o END do seu corpo. Embora facilite a implementação provoca limitações que podem inviabilizar certas aplicações

- O uso de Interface Modules para comunicação entre processos possui um funcionamento parecido com o monitor do Pascal. Leva 'a uma certa inflexibilidade no escalonamento de tarefas provocando restrições ao uso

- A utilização somente de Interface Modules para comunicação restringe a aplicação da linguagem

- Não implementa manuseio de situações de exceção.

Algumas vantagens importantes podem ser notadas na linguagem modula:

- E' uma linguagem bem modular o que facilita a programação levando 'a segurança e integridade

- Permite a programação em todos os níveis do sistema, requisito importante para programação de sistemas operacionais

- Uso de Modules devices para implementar acesso 'a I/O. Permite ao programador escrever procedimentos para periféricos na própria linguagem

- Devido 'a modularidade (Processos, Interface Modules, Device Modules) e 'a forma como e' implementada a comunicação permite a verificação em tempo de compilação de muitos erros difíceis de detectar em tempo de execução. Isto leva 'a segurança, integridade do sistema programado.

c) ADA

Na avaliação da linguagem ADA podemos verificar algumas restrições:

- So' oferece um tipo de comunicação através do conceito de rendez-vous, que embora seja um conceito simples não oferece alternativa. O método síncrono de transmissão de dados requer interações sucessivas com o escalador o que provoca perda de tempo

- O método síncrono de transmissão provoca 'as vezes espera desnecessária por parte do enviado. Este fato poderia ser

resolvido através da criação de TASKS intermediárias, mas que poderiam ter um custo elevado no caso de serem precisas muitas

- Política de escalonamento não acessível ao programador
- A falta de facilidades de baixo nível para o controle da proteção de dados compartilhados contra acesso concorrente. O único meio de controle para esse tipo de problema é a ENTRY CALL. Esse conceito provoca sobrecarga em relação a sistemas implementados por região crítica devido ao tamanho e interações necessárias com o escalador
- Um certo determinismo na comunicação entre TASKS. Não permite a possibilidade de se esperar por um conjunto de ENTRY CALLs
- Não permite a iniciação de mais de uma instância da mesma TASK ao mesmo tempo. Decorre da ambiguidade do nome da ENTRY de uma TASK com várias encarnações simultâneas.

Podemos verificar algumas vantagens na linguagem ADA:

- O conceito "rendez-vous" é elegante e simples de implementar
- A comunicação é assimétrica, ou seja, a TASK chamada não precisa saber o nome de quem a chamou e permite a troca de dados nos dois sentidos dentro do rendez-vous
- Facilidade para implementação de TIME OUT através do comando DELAY evitando bloqueio perpétuo.
- Permite compilação separada, facilitando a execução de grandes projetos
- Flexibilidade no controle das tarefas. Permite declaração de TASK dentro de outra TASK sendo mantido um controle entre as duas (PAI e descendente). Permite ativação de uma TASK do programa principal. Uma TASK pode pedir o seu aborto.

d) CHILL

A partir de uma avaliação da linguagem CHILL no que diz respeito a concorrência, podemos verificar algumas restrições:

- Não possui operações para o controle de E/S, interrupções, etc. Este fato dificulta a implementação de um sistema na linguagem CHILL em todos os níveis
- Devido ao fato acima não é uma linguagem com atributos para uso geral
- Linguagem destinada ao especialista de software devido à

complexidade da aplicação 'a qual e' voltada (em aplicações específicas este problema pode ser superado pela escolha de um sub-conjunto da linguagem).

A linguagem CHILL apresenta algumas vantagens entre as quais podemos citar:

- Oferece uma gama variada de instrumentos para o controle de concorrência, sendo neste aspecto uma linguagem muito flexível
- Linguagem modular, o que facilita a programação de grandes projetos (segurança)
- Fornece operações padronizadas de manuseio de condições de exceção ("over-flow", endereçamento, etc.)
- Flexível no controle de processos. Processos podem ser ativados por comandos em diferentes pontos e um processo pode provocar sua terminação
- Podem coexistir diferentes encarnações de um mesmo processo (instâncias).

IV.2. ESCOLHA DA LINGUAGEM BASE

Pela avaliação apresentada podemos verificar que todas as linguagens apresentam aspectos positivos e aspectos negativos em relação aos requisitos apresentados anteriormente. Na verdade nenhuma das linguagens apresentadas consegue responder de maneira totalmente satisfatória à todos os requisitos apresentados para uma linguagem com aspectos de concorrência de uso geral.

Torna-se necessário restringir os requisitos e também a complexidade exigida da linguagem através da definição da aplicação do sistema. A definição da aplicação facilita a escolha de uma linguagem, pois torna certos requisitos fundamentais e outros opcionais. À esses requisitos fundamentais nem todas as linguagens serão capazes de responder satisfatoriamente, sendo que a escolha e' feita sobre a linguagem que melhor se adapte 'a aplicação.

Reduzindo o campo de aplicação para sistemas de tempo real (notadamente telecomunicações) com multiprocessamento distribuído, podemos de início eliminar as linguagens Pascal Concorrente e Modula devido, principalmente ao fato das duas terem sido especificadas basicamente para utilização acadêmica e projeto de sistemas operacionais respectivamente e fazerem uso de memória compartilhada (dados) para comunicação entre processos

(processadores).

As duas linguagens restantes, ADA e CHILL, foram desenvolvidas para aplicação no sistema acima mencionado. Possuem pontos comuns e pontos divergentes nos quais uma supera a outra, porém, tres aspectos levam 'a escolha da linguagem CHILL como base. O primeiro aspecto, mais importante, e' que a linguagem CHILL sera' a linguagem padrão para as telecomunicações, sendo que as telecomunicações são o objetivo final deste sistema aqui proposto. O segundo aspecto e' a gama variada de instrumentos para o controle de concorrência existentes na linguagem CHILL, ao contrário da ADA que so' tem um instrumento (Rendez-vous). Este aspecto e' muito importante como visto anteriormente pois oferece flexibilidade 'a aplicação. O terceiro são alguns conceitos desta linguagem, tais como RECEIVE CASE e instâncias, muito úteis na implementação de sistemas telefônicos.

E' importante ressaltar que a linguagem CHILL não e' a melhor em termos gerais, mas sim a que melhor se adapta pelos motivos apresentados 'a aplicação desejada.

IV.3. APRESENTAÇÃO DOS ASPECTOS DE CONCORRÊNCIA A SEREM IMPLEMENTADOS

A seguir definem-se os recursos de concorrência 'a disposição do usuário. Estes recursos serão implementados na forma de primitivas baseadas em parte nos aspectos de concorrência definidos no CHILL através de um sub-conjunto desta linguagem e uma outra parte em novos aspectos a serem definidos e não cobertos na linguagem acima mencionada ou sugeridos de outra maneira. Os aspectos a serem definidos tem de levar em conta a aplicação 'a qual o núcleo se destina. E' importante ressaltar que na indisponibilidade de um Compilador CHILL o núcleo sera' descrito nas linguagens PL/M e ASSEMBLER. Os programas do S.O e usuário serão descritos em PL/M.

IV.3.1. Sub-conjunto da Linguagem CHILL

Foi escolhido um sub-conjunto que fosse abrangente, ou seja, cobrisse a maior parte possível dos aspectos de concorrência e fornecesse flexibilidade 'a aplicação. Outro objetivo foi a simplificação a fim de evitar transtornos nesta primeira implementação.

a) Ativação e desativação de processos

Ativação de um processo

START PROCESS - ativa um processo

Um processo determinado pode ativar qualquer outro processo do mesmo programa. O processo principal de um programa e' ativado através de um START implícito gerado pelo núcleo.

Desativação de um processo

STOP PROCESS - desativa um processo

O STOP so' pode ser acionado pelo próprio processo. A desativação pode ocorrer através do STOP ou através de um ABORT ocasionado por outro processo do mesmo programa.

b) Sincronização de processos e comunicação entre processos

Estes dois aspectos de concorrência serão tratados, nesta proposta, por meio das mesmas ferramentas visando uma simplificação e otimização do núcleo.

Tanto a comunicação como a sincronização de processos serão realizadas por meio de sinais.

O conceito de sinal a ser implementado tenta cobrir uma gama variada de possibilidades em termos de concorrência visando atender a vários requisitos da aplicação. Essa proposta pelos aspectos mencionados ultrapassa, em termos de aplicação, as características da proposta de sinal contida na linguagem CHILL.

Quatro operações serão realizadas no que diz respeito aos sinais: SAVE SINAL, LIBERA SINAL, SEND SINAL e RECEIVE SINAL.

SAVE SINAL sera' utilizada pelo usuário para evitar que um sinal a ele enviado seja liberado ao final de sua execução. Todos os sinais destinados a um usuário e não salvos são liberados ao final de sua execução.

LIBERA SINAL sera' utilizado pelo usuário para liberar de forma expressa um sinal anteriormente enviado por ele mesmo. Haverá liberação automática pelo núcleo de todos os sinais alocados a um processo e não utilizados pelo mesmo processo ao final de sua execução.

SEND SINAL sera' utilizada pelo usuário para o envio de mensagens visando a comunicação ou seja a sincronização com outros processos.

RECEIVE SINAL - utilizada pelo usuário para pedir o recebimento de sinais a ele destinados.

Cabe ao núcleo a gerência e controle dos sinais no sistema, protegendo o sistema quanto ao uso indevido das primitivas acima, pelo usuário.

O processo usuário pode pedir a alocação ou liberação de dois tipos de sinais distintos: sinal longo e sinal curto.

O sinal longo, com dados até 64 bytes, é utilizado basicamente para comunicação, enquanto o sinal curto, sem dados e com 14 bytes, é usado principalmente para sincronização.

Também será permitido ao usuário informar ao núcleo no pedido de SEND ou RECEIVE o modo de comunicação desejada: síncrona ou assíncrona. O usuário também determina se a comunicação será na forma difusão ou para um processador específico no caso de sinal externo. Essa última diferença é transparente para o núcleo sendo determinada através do hardware do circuito de comunicação.

A estrutura de sinais tenta da forma proposta atender ao máximo em flexibilidade à aplicação sem complicar em demasiado a implementação do núcleo.

Dois motivos principais e ligados um ao outro levaram a não implementação de buffers. O primeiro é que tudo o que é realizado por meio de buffer pode também ser realizado por meio de sinais como proposto, de forma transparente ao usuário. O segundo motivo é que isto simplifica a implementação.

c) Exclusão mútua

É um conceito importante quando aplicado aos sistemas com memória compartilhada e/ou co-processamento. A exclusão mútua nesse sistema é garantida pelas ferramentas de concorrência já implementadas (sinais, etc.).

IV.3.2. Outros Aspectos de Concorrência não Encontrados no CHILL e Implementados

Certas necessidades inerentes à aplicação ou necessárias e não propostas diretamente no CHILL são aqui descritas visando a sua implementação.

a) Temporização de Processos

Através de um esquema de temporização um processo pode pedir o seu atraso por um período especificado de tempo evitando bloqueio perpétuo e fornecendo flexibilidade ao sistema. A temporização pode ser cancelada se o processo assim desejar (no caso da ocorrência de um evento antes do fim da temporização). Ao findar a temporização o processo e ela relacionado e' ativado.

Temos 3 tipos de tratamento de tempo com relação a processo: saída por tempo, temporização cíclica e temporização comum.

Saída por tempo - para evitar bloqueio perpétuo. Utilizada pelo processo quando o mesmo fica esperando receber um sinal ou quando um sinal por ele enviado seja recebido. (Comunicação síncrona).

Comum - o vencimento da temporização ocorre apenas uma única vez. Deve haver novo pedido para nova ocorrência da temporização.

Cíclica - o vencimento da temporização ocorre a intervalos regulares e especificados no pedido de temporização. Deve ser feito um pedido de cancelamento para cessar os vencimentos desta temporização.

São duas operações:

PEDETEMP () - pede temporização
 CANCTEMP () - cancela temporização

Os parâmetros entre parênteses definem o tempo de temporização, o tipo (cíclica ou comum), processo, etc.

b) Tempo de execução de um processo

O valor do WDT sera' passado quando da criação do processo. O processo que executa a primitiva START criando o outro processo fornece também o WDT desse processo. Este esquema fornece flexibilidade ao sistema. Todos os processos de iniciação possuem um WDT default definido pelo sistema para sua execução.

c) Controle de interrupções e manuseio de condições de exceção no núcleo

Devem existir testes realizados pelo núcleo para garantir o bom funcionamento do mesmo e do hardware básico do módulo local sob controle do núcleo.

Deve ser provida uma área de memória inviolável a fim de se

armazenar dados que devem ser salvos quando houver reiniciação do módulo.

O núcleo deve salvar todos os registros nos quais ele for mexer.

Será implementado um controle de interrupções (inibe, habilita, tratamento) e manuseio de condições de erro (erro de ponteiros, erros nos pedidos ao núcleo, "over-flow", etc.). Deve ser possibilitado ao usuário o tratamento de certos tipos de falha, provocadas por seus processos, quando o mesmo desejar.

CAPÍTULO V

ESTRUTURA DO NÚCLEO E SUAS FUNÇÕES

V.1. FUNÇÕES DO NÚCLEO

Em sistemas de tempo real, tais como telefonia, a multiplicidade de tarefas e recursos a controlar, assim como a escassez dos mesmos torna necessária a existência de um programa que coordene as tarefas e forneça um meio seguro e controlado de acesso aos recursos.

Esse programa é chamado núcleo (também pode ser chamado de executivo, supervisor, monitor) e controla os recursos de CPU, memória, timers, comunicação entre processadores, interrupções.

Um sistema com multiplicidade de tarefas competindo por recursos e sem um núcleo teria um diagrama do tipo da Figura (V.1), enquanto que um sistema com núcleo teria um diagrama igual ao da Figura (V.2).

Pelas figuras podemos verificar que o sistema representado na Figura (V.2) é muito mais simples de implementar, depurar e manter, pois as suas interfaces são bem mais estabelecidas e o controle dos recursos é centralizado em um único bloco (núcleo).

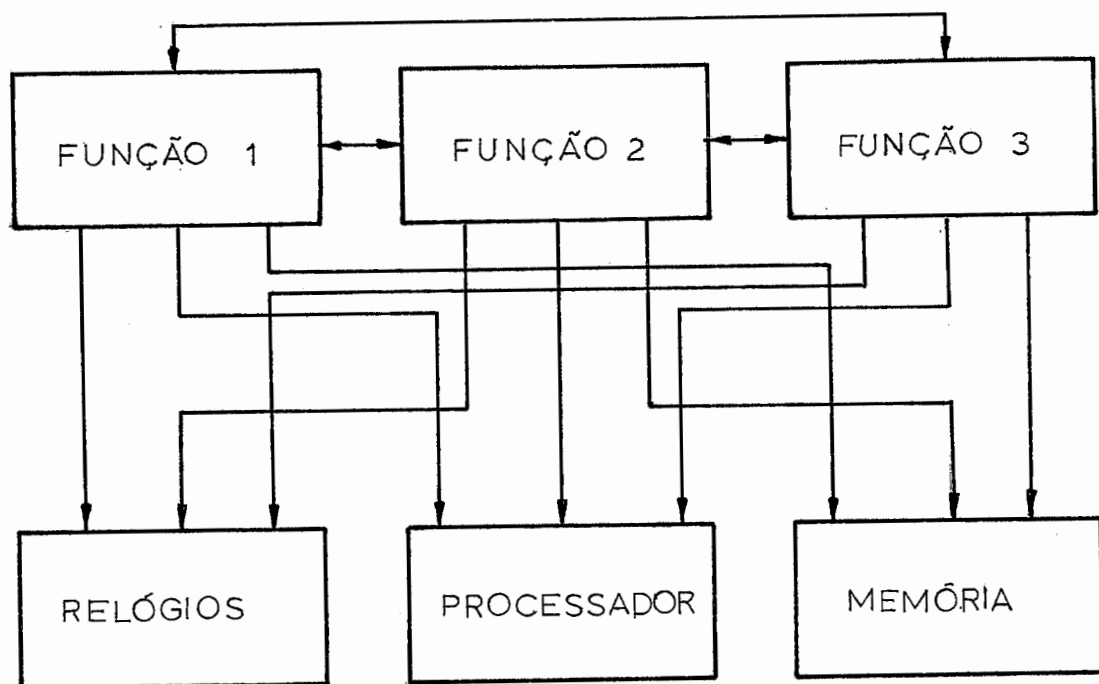


FIGURA (V.1)

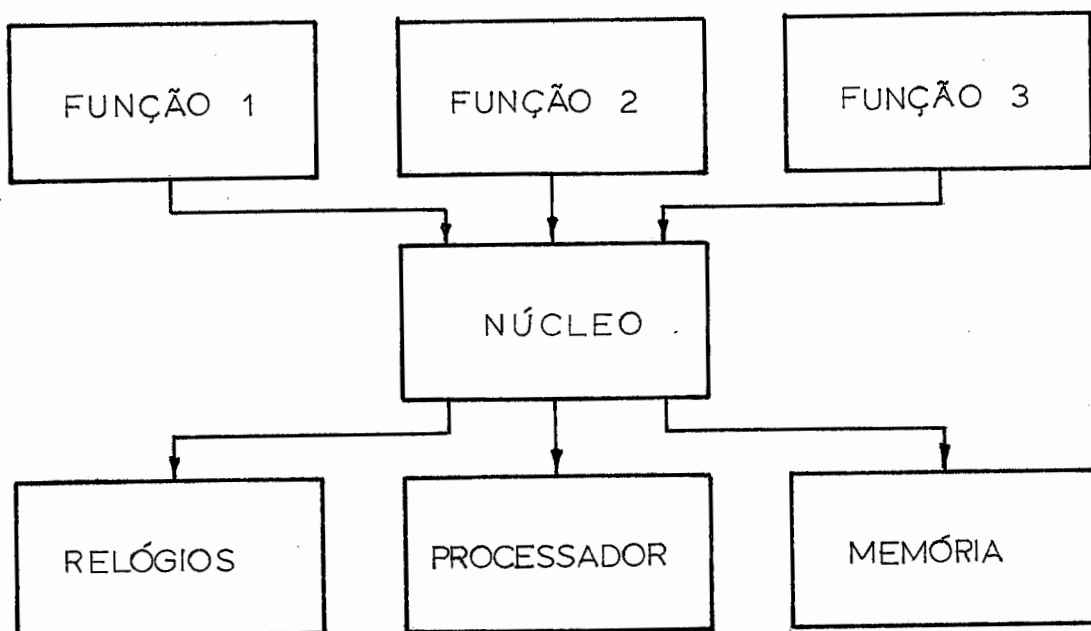


FIGURA (V.2)

O núcleo é portanto a parte do sistema operacional responsável pelo controle e gerência dos recursos do processador, possibilitando a sua utilização concorrente por vários processos. Para realização de seus objetivos o núcleo é formado por um conjunto de blocos interdependentes e cada qual responsável por uma função de controle de recursos e/ou de controle do funcionamento do processador local.

As funções, das quais é constituído o núcleo, se relacionam entre si e podem ser subdivididas em funções menores.

O núcleo é formado pelas seguintes funções: Iniciação, testes, controle de sinais, controle da comunicação externa, controle das interrupções, controle de falhas, gerência de processos e controle de temporização.

V.1.1. Diagrama de Interação das Funções

O diagrama apresenta as diversas funções do núcleo e suas relações entre si e com o mundo externo ao núcleo (S.O., usuário, hardware). Figura (V.3).

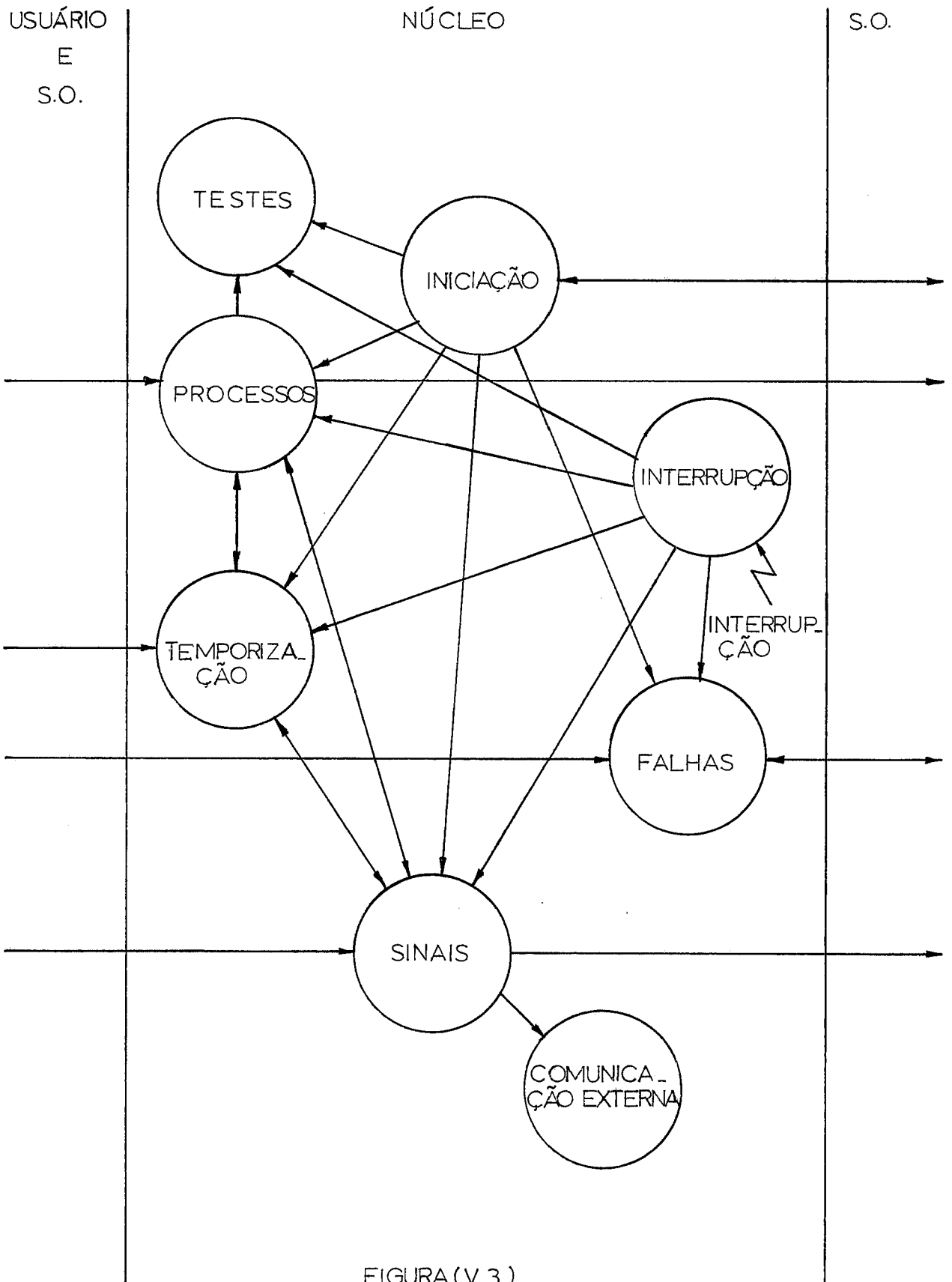


FIGURA (V. 3)

V.1.2. Descrição das Funções

a) Função de Iniciação

Esta função tem como objetivo permitir o posterior funcionamento, de modo correto, do processador local e seus processos.

Pode ser ativada por meio de uma chamada do processo tratador de falha do usuário ou S.O., pela função controle de falhas (falha que provoca reinicialização), pelo configurador ou depurador.

Chama todas as demais funções do núcleo.

Após ser ativada, esta função inibe interrupções, inicia a pilha do núcleo, verifica a existência do depurador via primitiva do configurador e chama cada uma das demais funções pedindo que a mesma inicie os seus dados internos e programe o hardware sob o seu controle. Cada função pode, nesse caso, retornar erro no caso de não conseguir programar o "hardware" corretamente (as que possuem). A primeira função a ser chamada pela iniciação é a função de testes. Esta função retorna erro no caso de problema em algum teste, levando a função iniciação a verificar o número de reiniciações sucessivas e não se iniciar (assinala alarme e entrega o controle ao depurador, caso o mesmo exista, no caso deste número ultrapassar o limite). Se não houver problemas na função de testes a iniciação chama a seguir as funções de controle de falhas, gerência de processos, controle de sinais e controle de temporização. No caso de todas se iniciarem corretamente a iniciação transfere o controle do processador para a função de controle de processos. O controle é passado ao gerenciador de processos e só retorna à iniciação no caso de reiniciação. As falhas na parte de testes provocam a parada do processador, as demais funções não provocam parada na iniciação. A função controle de comunicação externa pode assinalar falha para o controle de falhas e o núcleo se inicia sem esse recurso que não conseguiu se iniciar corretamente.

Função:

- Iniciar processador local

Chamada por :

- Transferência de controle do Configurador
- Controle de falhas (falhas que provocam reiniciação - Primitiva REINIPR).
- Comando do Depurador

Chama :

- Demais funções do núcleo para se iniciarem (testes processos, falhas, temporização, sinais)
- Configurador (Primitiva INFDEP)
- Controle de falhas (Primitiva PARAPRO)
- Transferência de Controle para função controle de processos

b) Testes

Esta função tem como objetivo a realização de testes no hardware sob controle do processador local (no qual o núcleo está residente) prevenindo o mal funcionamento do mesmo. Os testes são realizados durante a iniciação, escalados pelo controle de iniciação, ou são realizados periodicamente no tempo livre do processador, escalados pelo controle de processos quando não há processo ativo a ser escalado, sendo que neste caso os testes são realizados por partes a cada chamada.

Os testes realizados provocam a parada do processador no caso de falha, levando o processador a assinalar um alarme e entregar o controle ao depurador (caso exista).

Os seguintes testes são realizados por esta função: Teste da CPU, teste de memória RAM, teste de memória EPROM, teste de timers, teste do circuito de proteção de memória, teste do circ. de controle das pilhas.

O teste de UCP é realizado através da execução de um conjunto de instruções cujo resultado final, conhecido, é comparado com o resultado obtido.

O teste de memória RAM é realizado através da escrita, leitura e comparação de um valor conhecido.

O teste de memória EPROM é realizado através de um check-sum realizado nesta memória e comparado com um resultado conhecido.

O teste dos contadores de tempo é realizado através da programação dos contadores e posterior leitura e comparação com resultado esperado após a execução de um procedimento conhecido.

O circuito de proteção de memória é testado a partir da sua programação e tentativa de acesso à uma área proibida.

O circuito de controle de pilha é testado a partir da provocação de um estouro de pilha do núcleo.

Função:

- Testar hardware do processador local

Chamada por:

- Controle de iniciação
- Gerência de processos
- Interrupções (WDT, VIOL PROT MEM, CONT PILHA)

Chama:

-

c) Gerência de processos

Esta função controla as filas de processos, os estados dos processos, a sua escalação e execução. Controla a criação dos processos, informa ao sistema operacional e recebe deste as informações necessárias para o funcionamento de ambos. Quando uma determinada falha ocorre nesta função ela assinala para o controle de falhas para que a mesma seja tratada.

A gerência de processos é chamado pelo controle da iniciação após a iniciação ter sido completada.

Após obter os dados do configurador e se o mesmo não acusar problemas, a função gerência de processos passa para a próxima fase de sua execução. Nesta fase ela escala os processos de iniciação de cada programa, controlando a sua execução.

Quando não existe processo ativo para ser escalado esta função chama a função de testes.

A função gerência de processos recebe o controle do processador, verifica se alguma condição foi satisfeita e coloca o processo atendido na fila de ativos se for o caso.

A seguir escala o primeiro processo da fila de maior

prioridade, programando antes o WDT de acordo com o valor pedido e transfere o controle do processador para o mesmo. Esta função é avisada quando um sinal é enviado para um processo do processador pelo controle de sinais ou quando chega um sinal externo. Também é avisada do fim de uma temporização pelo controle de temporização. Esta função pode colocar um processo de prioridade mais baixo como suspenso interrompendo sua execução, para escalar um processo de prioridade mais alta. Ao final da sua execução, o processo devolve o controle do processador a esta função para que a mesma escale outro processo. Esta função controla também a criação e desativação de processos e assinala ao sistema operacional estes fatos.

Todo processo termina sua execução por um STOP, REATIVA ou PEDIDO DE ESPERA. Os processos de tratamento de falha sempre terminam com um REATIVA ou STOP enquanto os processos de iniciação terminam sempre com um STOP.

Função:

- Controlar processos em execução no processador local

Chamada por:

- Controle de iniciação
- Controle de Temporização
- Controle de Sinais
- Controle de Falhas
- Interrupção relógio de tempo real (4ms)
- Usuário (via primitivas - STARTPR, STOPROC, etc).

Chama:

- Configurador (primitiva INICONF)
- Controle de testes
- Controle de falhas
- Sistema operacional (primitiva INFPRO)
- Controle de sinais
- Controle de Temporização

d) Controle de sinais

Esta função controla todos os recursos do núcleo que dizem

respeito aos sinais. Oferece ao usuário a possibilidade de liberar sinais, enviar sinais, receber sinais nos modos permitidos. Possibilita a comunicação e sincronização de processos no sistema. Controla o número de sinais alocados por um mesmo usuário durante a sua execução, não permitindo que o mesmo aloque mais de 10 sinais por execução. Libera todos os sinais alocados a um processo que acaba de executar e que não foram consumidos, a menos que os mesmos estejam marcados para SAVE. Controla a colocação dos sinais na fila externa para que sejam enviados pelo controlador de comunicação externa. Obtem sinal externo recebido e o coloca em sua fila de prioridade. Quando um processo envia um sinal e fica esperando que o mesmo seja recebido, ou quando pede recepção de sinal e fica na espera, o bloco de controle de sinais aciona a função de controle de temporização a fim de que a mesma realize uma temporização (time-out) para o processo, evitando bloqueio perpétuo.

O bloco de controle de sinais sinaliza para a gerência de processos sempre que o processo em execução envia um sinal, recebe um sinal ou requisita envio ou recepção de sinal com espera e a mesma acontece. Quando um processo tenta liberar ou enviar e receber um sinal inexistente, uma falha é sinalizada para o bloco de controle de falhas.

A função de comunicação externa é uma função escrava desta função.

Função:

- Gerenciar as comunicações por sinal do processador local

Chamada por:

- Controle de iniciação
- Gerência de processos
- Usuário (via primitivas - SENDSIN, RECESIN, etc)
- Interrupção de recepção de sinal externo
- Interrupção de relógio de tempo real (4 ms)
- Interrupção de relógio de 1 seg

Chama:

- Controle de temporização
- Gerência de processos
- Controle de comunicação externa
- Controle de falhas

- Sistema operacional (primitiva PRODES)

e) Controle de temporização

Nesta função estão implementados todos os recursos de tempo do núcleo. Controla os pedidos de temporização, gerencia as células de temporização e sinaliza ao bloco de controle de processos quando vence uma temporização para determinado processo. Por sua vez, o bloco de gerência de processos avisa a esta função quando um processo é desativado a fim de que as células e temporizações à ele ligadas sejam liberadas. No caso de haver pedidos de acionamento, cancelamento ou no caso da gerência de processos informar desativação de um processo que deveria ter uma temporização e não tem, o controle de temporização armazena a informação no controle de falhas e chama a função controle de falhas.

Função:

- Gerência as bases de tempo do processador local

Chamada por:

- Controle de iniciação
- Controle de sinais
- Usuário (primitivas - PEDTEMP, CANTEMP)
- Interrupção de relógio de TEMPO REAL (4 ms)
- Interrupção de relógio de 1s

Chama:

- Controle de falhas
- Gerência de processos

f) Controle de falhas

Esta função gerencia o nível do tratamento de falhas e o modo de tratamento das falhas. É também responsável pela manutenção de uma área de memória onde estão armazenados os dados da última falha ocorrida e que podem ser acessados pelo S.O. ou usuário (processo tratador de falhas).

Possui interfaces com as funções de controle de processos, temporização e sinais do núcleo. A área de armazenamento dos

dados não e' apagada.

Função:

- Gerencia o modo de tratamento das falhas ocorridas no processador local.

Chamada por:

- Controle de processos
- Controle de iniciação
- Controle de sinais
- Controle de temporizações
- Usuário (via primitivas - ALTENIV, etc)
- Sistema Operacional (via primitivas)
- Interrupções (WDT, violação prot. memória, controle de pilha)

Chama:

- Controle de iniciação (primitiva - REINIPR)
- Depurador (caso exista e o modo indique parada)
- Sistema Operacional (primitiva - INFALSO)
- Gerência de processos
- Configurador (primitiva - INFMOD)

g) Controle de comunicação externa

Esta função e' responsável pelo controle do circuito de comunicação externa do processador (bloco de comunicação). Trabalha como uma função escrava da função controle de sinais.

Esta função e' acionada pelo controle de sinais para enviar ou receber sinais externos. Cabe a essa função a programação do hardware necessário para o funcionamento do circuito de comunicação externa e todo o controle de seu funcionamento. Verifica o resultado da operação e indica falha, se a mesma ocorrer, para o controle de sinais.

Função:

- Controla o circuito de comunicação externa do processador local.

Chamada por:

- Controle de sinais

Chama:

-

h) Controle de interrupções

Na verdade não se trata de uma função e sim de um bloco onde estão implementadas a programação e o atendimento das interrupções necessárias ao funcionamento do núcleo. Após atender a interrupção este bloco desvia para a função que deve atendê-la. Existem interrupções de relógio, de recepção de sinal externo, de falha (condições de exceção, de WATCH DOG TIMER).

Função:

- Fornecer as rotinas para tratamento das interrupções do sistema.

Chamada por:

- Interrupções (relógio 4ms, relógio 1s, recepção sinal externo, violação de PROT MEM, estouro de WDT, estouro de pilha e depurador).

Chama:

- Cada função que trata a interrupção mencionada.

V.2. ESTRUTURA DO NÚCLEO

O núcleo é um programa constituído de três partes distintas: bloco comum, bloco das funções e bloco das interrupções.

a) Bloco Comum

No bloco comum encontram-se os dados de acesso externo ou comuns a todo o núcleo.

a.1) Área de definição do núcleo

Nesta área está a descrição do núcleo, suas funções, sua aplicação, características (tamanho de código e dados, linguagem e processador), versão (data, autores).

a.2) Área de comunicação do núcleo

Os dados do núcleo utilizados externamente por outros programas tais como os buffers de comunicação (BUFCOM1, ...) e as primitivas, assim como os dados externos utilizados pelo núcleo (primitivas do Sistema Operacional), biblioteca de sub-rotinas e macros, estão declaradas nessa área. Também está declarado nessa área o bloco de comunicação global.

a.3) Área de dados do núcleo

As variáveis do núcleo (células de sinais, células de temporização, etc.) e constantes de uso geral são declaradas nessa área. Nesta área estão também as sub-rotinas e macros de uso exclusivo do núcleo e compartilhadas por mais de uma função.

b) Bloco das Funções

Este bloco é constituído de 6 partes, cada uma referente à uma função do núcleo. Cada função possui as seguintes áreas:

b.1) Área de definição da função

Esta área contém a descrição do funcionamento da função e seus recursos.

b.2) Área de comunicação da função

Contém os dados desta função utilizados por outras funções do núcleo e os dados de outras funções que são referenciadas nesta. Permite montagem em separado de cada função.

b.3) Área de dados da função

As variáveis e constantes da função são declaradas nessa área. Nesta área se encontram também as sub-rotinas e macros de uso exclusivo dessa função.

b.4) Área de comandos

Contém o algoritmo de execução da função

Tanto no bloco comum, como no bloco das funções qualquer ítem não existente deve ser assinalado com um comentário ----" N.E.

c) Bloco das Interrupções

Este bloco é constituído pelas rotinas de tratamento de cada tipo de interrupção reconhecida no sistema.

V.3. BLOCO DE COMUNICAÇÃO GLOBAL

Arquivo único que contem definições utilizadas na compilação de programas e do núcleo.

- identidade de todos os sinais do sistema
- nome dos programas do sistema
 - nome dos processos do programa
 - nome de cada instância do processo
- macros e sub-rotinas
- endereços "hardware" (básico).

Início da Área de RAM, fim da Área de Ram, End do Alarme, etc.

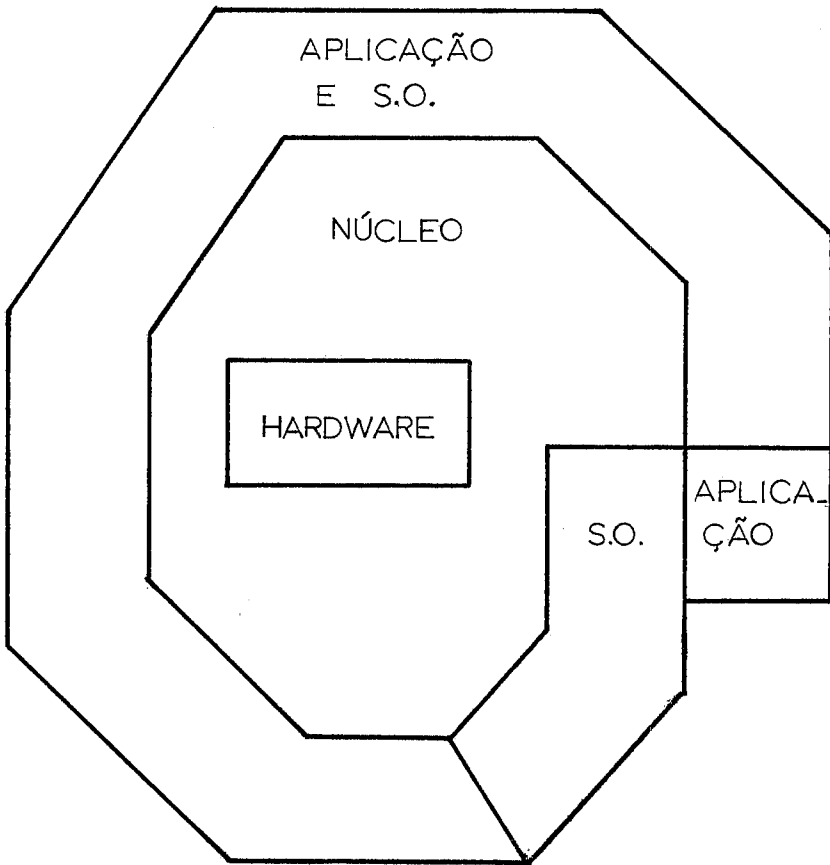
- Nome das primitivas do configurador e S.O. acessadas externamente (declaradas como externas).
- Nome das primitivas do núcleo (declaradas como externas)

O número do processador e' obtido pelo configurador a partir da leitura de uma porta "hardware".

V.4. INTERFACES DO NÚCLEO

Para uma melhor descrição do núcleo e suas interfaces, sera' utilizado o metodo de descrição do mesmo por camadas (níveis). Desta forma temos 3 níveis nos quais podemos dividir o núcleo:

Interface física, interface com o sistema operacional e interface com a aplicação . Figura (V.4).



MÓDULO LOCAL
(PRIMITIVAS)

MÓDULOS
EXTERNOS
(SINAIS)

FIGURA (V. 4)

a) Interface Física

A interface física compreende o relacionamento do núcleo com o "hardware" sob seu controle direto. Esse "hardware" é constituído basicamente de duas partes principais: módulo local de controle e módulo local de comunicação, que fazem parte do módulo local básico. Figura (V.5).

Será feita uma descrição breve da interface.

a.1) Interface com o módulo local de controle

O módulo local de controle é constituído pela UCP, memória, circuito de proteção de memória, timers e relógios, controle de interrupção e demais circuitos ligados à essas partes mencionadas. Cabe ao núcleo o controle e programação desse "hardware" a fim de assegurar o bom funcionamento de todo o sistema à ele ligado.

O bom funcionamento dessa parte do sistema é imprescindível para a continuação da operação do módulo local.

Dois tipos de ações são realizadas sobre esse "hardware": programação e testes.

Os testes visam assegurar que os circuitos mencionados estejam funcionando corretamente. Eles são realizados na iniciação ou reiniciação do núcleo e durante o tempo livre de UCP. Estes testes consistem do teste de UCP, teste de memória RAM, teste de memória ROM, teste dos relógios e teste do circuito de proteção de memória. Devido à importância desses circuitos quando ocorre falha em um desses testes, o mesmo é repetido 2 vezes. Se a falha persistir, os dados da mesma são armazenados em uma área de memória RAM para esse fim (não é apagada na iniciação) e o núcleo comunica ao sistema operacional a impossibilidade de continuar operando e inicia uma saída por tempo para espera da resposta. Se não for possível avisar ao sistema operacional ou se a resposta não chegar a tempo, o núcleo aciona o alarme no módulo local e transfere o controle ao depurador (o controle só retorna ao núcleo com RESET, POWER-ON ou por comando GO do depurador). Após receber a mensagem do núcleo o sistema operacional armazena a falha em memória de massa, aciona o alarme, envia mensagem aos demais processadores comunicando o fato e bloqueia o processador local.

O depurador é um programa operado a partir de um terminal de vídeo e que permite a depuração do processador local através da implementação de comandos para leitura/escrita em registros e

memória, execução passo a passo, etc.

A programação do hardware do módulo local básico é executada sempre que o núcleo tem o controle do processador e vai ativar uma nova tarefa. Essa programação protege o sistema de eventuais falhas e permite ao núcleo obter o controle do processador nesses casos. Consiste da programação dos relógios de tempo real e WDT, da programação do controle de interrupções e da programação do circuito de proteção de memória.

a.2) Interface com o módulo de comunicação

Essa interface consiste dos testes e programação do hardware do módulo local de comunicação, responsável pelo envio e recepção de sinais externos ao módulo local. Os testes são realizados pelo microprograma do módulo local de comunicação a partir de um comando do núcleo.

Os testes são realizados na iniciação/reiniciação do núcleo e/ou no tempo livre de CPU. No caso de falha nos testes o módulo de comunicação avisa ao núcleo. Este tenta mais 2 vezes o teste e no caso de persistir a falha, bloqueia as comunicações externas para os usuários com exceção do sistema operacional. A seguir o núcleo comunica ao sistema operacional o bloqueio, devido a falha, da comunicação externa. O sistema operacional aciona o alarme no módulo local, armazena a condição de falha em sua base de dados e se possível em memória de massa.

A seguir o sistema operacional ativa um procedimento de comunicação externa que realiza tentativas periódicas de utilização do circuito, avisando ao núcleo para desbloquear a comunicação externa quando tiver sucesso em uma comunicação.

A programação do módulo local de comunicação é realizada durante a fase de iniciação/reiniciação do núcleo. Ela permite a utilização correta desse módulo, prevenindo contra eventuais falhas. O núcleo deve sempre manter o módulo de comunicação programado para receber e/ou enviar sinais. No caso de falha no envio de um sinal o núcleo tenta enviá-lo mais 2 vezes. Caso a falha persista é seguido o mesmo procedimento do caso de testes.

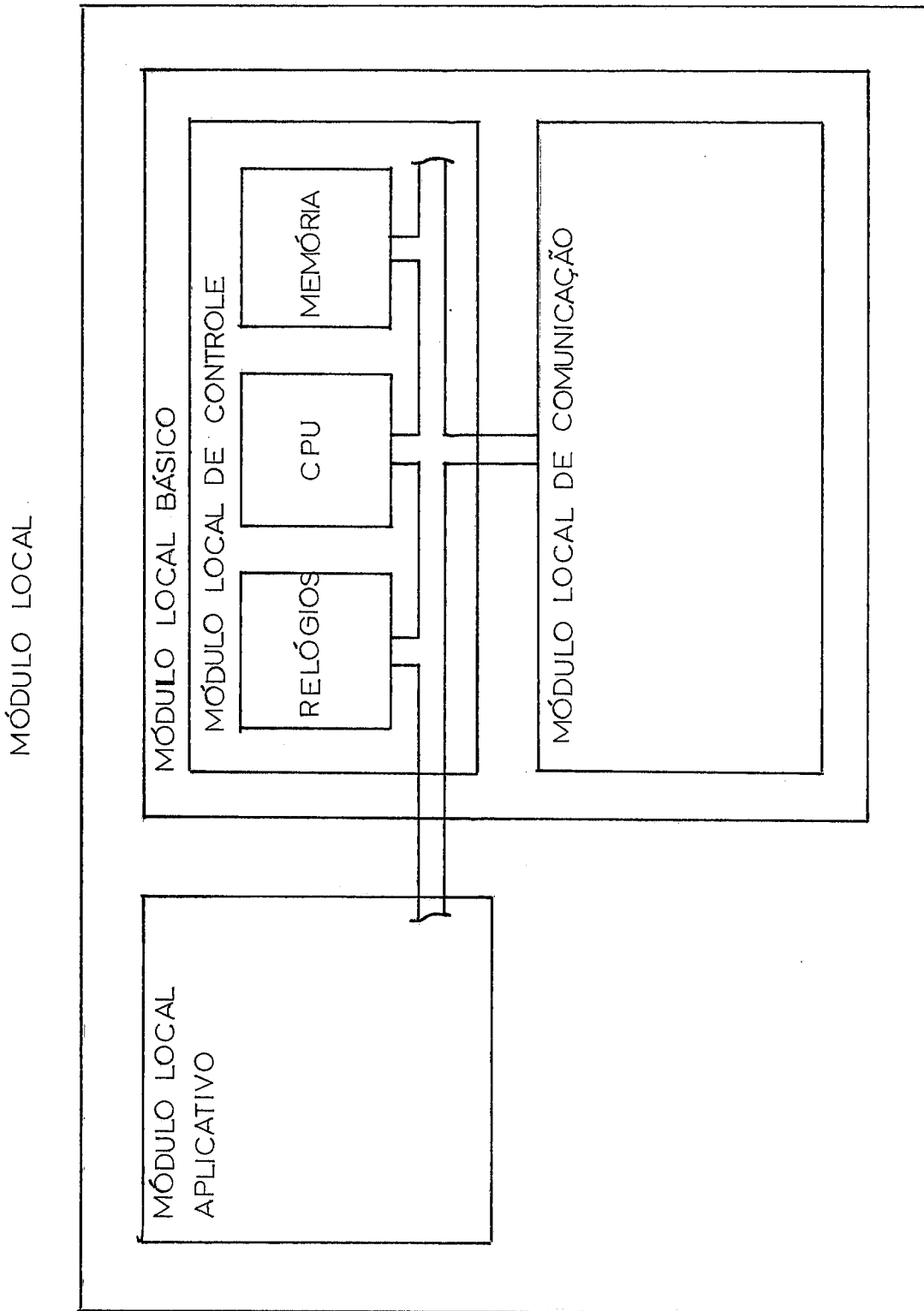


FIGURA (V5)

b) Interface com o Sistema Operacional e Configurador

A interface do núcleo com o sistema operacional ou configurador é implementada através de primitivas que são chamadas pelo núcleo. Essas primitivas são de uso exclusivo do núcleo para as suas comunicações e não podem ser usadas pelo usuário. Desta forma ao haver uma chamada de uma primitiva do tipo restrito acima, o núcleo verifica a prioridade do processo que está executando no momento (PRIPROEX), assinalando falha e passando o controle para a função controle de falhas.

A passagem de dados é feita através de BUFNUC. Este buffer é uma área de memória de 80 bytes utilizada pelo núcleo nas suas chamadas de primitivas do sistema operacional ou configurador para fornecer e receber dados de resposta. No caso de haver uma chamada de primitiva do S.O. de uso geral ou restrito, o S.O. reconhece quem chamou e desta forma também o buffer de comunicação através de PRIPROEX (se PRIPROEX = 5 é o núcleo que chama).

São mencionadas agora somente as primitivas da interface do S.O. com o núcleo necessárias para o funcionamento do núcleo (as demais são descritas de maneira sucinta no anexo II).

O S.O. ou configurador pode ser acessado por processos de outros processadores somente através de sinais.

c) Interface com a Aplicação

A interface do núcleo com a aplicação é implementada por meio de primitivas de uso geral, ou seja, que podem ser acessadas pela aplicação ou pelo sistema operacional. Desta forma qualquer usuário pode solicitar os recursos de gerência de processos, gerência de sinais, gerência de temporizações, etc. do núcleo.

A passagem de dados nestas primitivas é feita através de buffers do núcleo.

Existe um buffer para cada tipo de processo e o núcleo identifica o processo que chamou, e desta forma o buffer, através da variável PRIPROEX.

São 5 buffers:

- Bufcom 0 - para processos de prioridade 0
- Bufcom 1 - para processos de prioridade 1
- Bufcom 2 - para processos de prioridade 2

Bufcom 3 - para processos de prioridade 3
 Bufcom 4 - para processos de prioridade 4

O usuário tem acesso somente a seu buffer, prevenindo dessa forma acessos indevidos a outras áreas do núcleo.

Os processos de programas aplicativos só podem ser acessados por meio de sinais tanto por processos internos ou externos ao processador.

V.5. ESTRUTURA DE UM PROGRAMA APLICATIVO

Um programa de aplicação ou do sistema operacional é constituído de processos. Todo programa tem um processo de iniciação e processos comuns. O processo de iniciação é escalado automaticamente pelo núcleo na iniciação. Cabe ao processo de iniciação ativar processos do programa e iniciar dados necessários à execução normal do programa.

Os processos de iniciação possuem prioridade 1 e WDT padrão de 1 seg, e são sempre os primeiros a serem declarados e os processos tratadores de falhas (se houver) são os últimos.

Um programa em sua estrutura possui dois blocos: bloco comum do programa e bloco dos processos do programa.

a) Bloco Comum do Programa

Neste bloco estão os dados e definições utilizados por mais de um processo e os dados públicos do programa e os dados externos utilizados no mesmo.

a1.) Definição do programa

Contem a descrição funcional do programa, seus processos (tipo), autor, versão, tamanho da área de código e dados, linguagem utilizada e processador hospedeiro.

a2.) Comunicação

Nesta área encontram-se declarados os dados externos utilizados pelo programa entre os quais podemos citar: as primitivas do núcleo e S.O., os buffers de comunicação (biblioteca de sinais, biblioteca de programas, processos e instâncias, sub-rotinas e macros de uso geral) e o bloco de comunicação global.

a3.) Dados do programa

Os dados variáveis e constantes do programa utilizados por mais de um processo se encontram nessa área. Contem também as sub-rotinas e macros utilizadas por mais de um processo do programa.

b) Bloco de Processos do Programa

Este bloco e' constituído de uma descrição para cada processo do programa. O processo de iniciação e' o primeiro a ser descrito. Os processos são descritos da seguinte maneira:

b1.) Definição do processo

A descrição funcional (tipo do processo), número de instâncias, prioridade e WDT do processo são descritas nessa área.

b2.) Comunicação do processo

Dados utilizados por outros processos do programa. Permite montagem em separado do processo.

b3.) Dados do processo

Os dados variáveis e constantes de uso exclusivo do processo, assim como macros e sub-rotinas são descritos e declarados nessa área.

b4.) Corpo do processo

Algoritmos de execução do processo.

Tanto no bloco comum como no bloco de processos qualquer item não existente deve ser assinalado com N.E. (não existente).

Um processo tem acesso somente à sua área de código, à área de dados do programa, à sua área de dados, à área de comunicação e à sua pilha de uso.

CAPÍTULO VI

O NÚCLEO, SEUS ELEMENTOS DE CONTROLE

E RECURSOS DE CONCORRÊNCIA

O núcleo descrito nesta tese destina-se principalmente, 'a aplicação em sistemas telefônicos. Este tipo de sistema possui particularidades que devem ser atendidas pelo núcleo e demais programas do sistema operacional, tais como, controle rígido de tempo, controle de configuração, controle de falhas, etc.

E' importante ressaltar que a comunicação entre qualquer programa aplicativo e o sistema operacional no mesmo processador e' feita via primitivas que tornam a comunicação simples e rápida. O núcleo e o sistema operacional se comunicam também por meio de primitivas. Outro aspecto importante a ressaltar e' o fato de que o sistema operacional, 'a exceção do núcleo, pode ser acessado por meio de sinais por programas de outro processador. Isto permite a utilização de um recurso controlado pelo sistema operacional local por programas em mais de um processador.

VI.1. ELEMENTOS DE CONTROLE DO NÚCLEO

Através desses elementos, que são tabelas, dados, constantes, etc., o núcleo pode controlar os processos residentes no módulo local, gerenciar o uso do processador e oferecer aos usuários os serviços anteriormente mencionados.

O principal objetivo do núcleo e' controlar as várias unidades executivas residentes no módulo local e tornar seguro e eficiente o compartilhamento dos recursos do módulo local colocados 'a disposição do usuário, tais como, processador, comunicação externa, etc. As unidades executivas recebem o nome de processos. Programa e' um conjunto de processos responsáveis por determinada função no sistema, sendo que um dos processos recebe o nome de processo de iniciação e tem como função ativar os demais processos do programa e dar a partida para sua execução. Desta forma cabe ao núcleo o controle sobre os processos e cabe ao sistema operacional o controle sobre os programas e suas condições de funcionamento e sobre o processador local e sua condição de operação.

VI.2. RECURSOS DE CONCORRÊNCIA

Os serviços do núcleo são os recursos e facilidades colocados à disposição da aplicação e do sistema operacional pelo núcleo.

Os recursos e facilidades do núcleo são disponíveis para a aplicação e sistema operacional por meio de primitivas.

Desta maneira temos um único meio de acesso simples, rápido e seguro ao núcleo. A cada chamada de uma primitiva do núcleo, o processo passa os parâmetros de entrada e recebe os dados de saída via um buffer determinado segundo sua prioridade. Nessa área o núcleo obtém os dados de entrada da primitiva e coloca os dados de saída, não alterando registro algum. Esta comunicação por meio de uma área do núcleo permite flexibilidade na implementação e alteração dos parâmetros de entrada e saída da primitiva.

Entre os recursos do núcleo estão a temporização, comunicação por meio de sinais, sincronização por meio de sinais, controle de processos residentes no processador, etc.

VI.2.1. Controle de Iniciação

Esta função é responsável pela iniciação do processador local. Pode ser ativada por um comando do configurador quando este, após configurar o processador local transfere o controle para essa função. Pode ser ativada por um comando do depurador (G) ou pode ser ativada através de uma primitiva de uso restrito que pode ser chamada pelo controle de falhas.

VI.2.1.1. Primitivas

A primitiva abaixo pertence ao núcleo e permite o acesso a essa função.

REINIPR - Esta primitiva permite ao controle de falhas pedir a reiniciação do núcleo. No caso, a chamada só é aceita se o núcleo estiver no modo de tratamento sem parada no depurador.

Entrada - Não há

Saída - Não há

A primitiva relacionada abaixo pertence ao configurador e é utilizada pelo núcleo na sua iniciação.

INFHARD - Esta primitiva do configurador informa a existência dos circuitos de proteção de memória, WDT, controle de pilha e comunicação externa.

Entrada - Não ha'

Saída - Existe ou não cada circuito .

VI.2.1.2. Área de Comunicação

A passagem dos dados nas chamadas das primitivas e' realizada através das áreas de comunicação relativas a cada tipo de processo e do núcleo (BUFCOMO, BUFCOM1, BUFCOM2, BUFCOM3, BUFCOM4 e BUFNUC).

VI.2.2. Gerência de Processos

A gerência de processos e' ativada após a iniciação do núcleo. Após chamar o configurador e obter o número de programas e o ponteiro para o processo de iniciação de cada programa, escala os processos de iniciação um a um, entrando em regime normal ate' que não haja mais processos criados no processador local. A cada interrupção de relógio de tempo real (4ms) ou após receber o controle do processador, de um processo o núcleo escala um outro processo ativo prioritário para executar, ou o processo suspenso, se não houver processos prioritários.

O processo e' a unidade executiva do processador local sob controle do núcleo. Em um processador podemos ter vários programas, cada programa e' constituído de um processo de iniciação e processos comuns. Cada processo comum pode, por sua vez, ter instâncias a ele associadas. As instâncias são encarnações de um mesmo processo. O processo de iniciação e' escalado automaticamente, enquanto que os demais são criados através de primitivas. Os processos aplicativos so' podem se comunicar com outros processos aplicativos por meio de sinais.

O sistema operacional e' formado por programas que so' diferem dos programas aplicativos pela prioridade de seus processos e por poderem ser acessados por primitivas.

VI.2.2.1. Diagrama de Estados de um Processo

Um processo pode assumir várias condições de funcionamento

durante a execução de programas em um processador. Figura (VI.1).

Parados - Todos os processos de todos os programas (iniciação) estão nesse estado na iniciação do núcleo e desta maneira, não são vistos pelo mesmo. Um processo só deixa este estado e é visto pelo núcleo, através de um comando START dado por outro processo do mesmo programa. Um processo pode chegar a esse estado somente através de um STOP executado por ele mesmo, ou através de um ABORT executado por outro processo de seu programa. Do estado parado um processo só pode atingir o estado ativo. Da mesma forma o estado parado pode ser atingido por um processo a partir de qualquer estado através da execução do STOP ou ABORT.

No estado parado um processo não existe para o núcleo e qualquer sinal que chega para o processo parado é abandonado. Após ser iniciado o núcleo executa cada processo de iniciação de cada programa do módulo local através de um START.

Ativos - Neste estado os processos estão prontos para executar. Um processo chega a esse estado de três maneiras. A primeira através de um START dado por outro processo e que desta maneira o retira (cria o processo) do estado parado. A segunda através do atendimento de alguma condição de espera que o tira do estado Espera. A terceira pela reativação pedida por ele mesmo. Do estado ativo um processo pode atingir o estado executando a partir da escalação do núcleo ou o estado parado por meio de um ABORT executado por outro processo do mesmo programa. Nesse estado um processo pode receber sinais e estes são colocados nas listas. Existem listas de processos ativos por prioridades e cada célula contém o número do programa, o número do processo, a posição de execução, etc.

Executando - Existe apenas um processo neste estado, a cada momento, no processador. Neste estado um processo pode requisitar recursos do núcleo, enviar sinais, etc. Ele chega a esse estado através de escalação do núcleo e sai dele por três motivos. Através de um STOP, reativação ou através de uma espera condicionada por algum evento. Desse estado o processo pode ir para parado, ativo ou para espera. Existe uma única célula de um processo em execução e esta célula possui o número do programa, o número do processo, prioridade e posição de início de execução, etc.

Espera - Um processo se coloca nesse estado à espera de algum evento. Esse evento pode ser uma temporização, o envio de um sinal (no caso do processo pedir recepção de sinal com espera), a recepção de um sinal (no caso de um processo pedir o envio com espera). Um processo chega nesse estado através de um pedido de espera pelo evento a partir do estado executando. Um processo é retirado desse estado e colocado pelo núcleo no estado ativo quando o atendimento da condição de espera. Um processo também pode sair desse estado para o estado parado por meio de um ABORT. Nesse estado um processo pode receber mensagens. Temos filas de processos à espera de temporização (organizadas por classes de

temporização), à espera de sinais, etc.

Suspenso - Um processo é colocado nessa condição se ao ocorrer uma interrupção de relógio houver um processo prioritário para executar. Quando não há mais processos prioritários o núcleo antes de verificar as listas de processos ativos recolhe o processo suspenso e o executa caso exista.

Erro - Um processo é colocado nesse estado quando provoca falha e o tratamento da mesma vai ser feito pelo próprio usuário. O processo só fica nesse estado durante a execução do processo de tratamento de falhas, sendo depois parado.

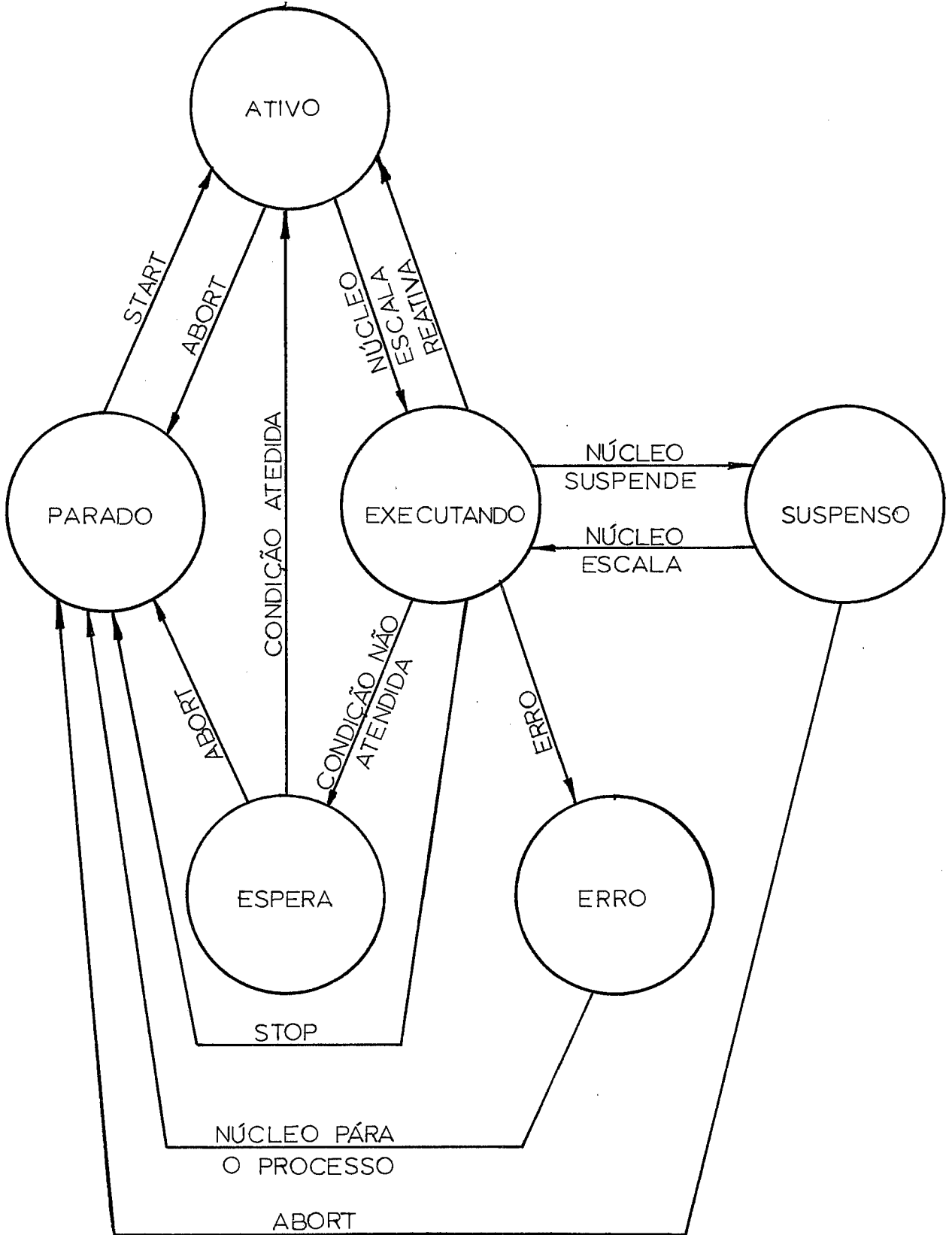


FIGURA (VI.1)

VI.2.2.2. Tipos de Processos

São cinco tipos de processos cada um ligado a uma prioridade de execução.

Processos tipo 0 - são processos tratadores de falha. Não podem ser suspensos.

Processos tipo 1 - são processos de alta prioridade do sistema operacional ou processos de iniciação. Podem interromper os processos tipos 3 e 4 e são prioritários aos processos tipo 2 na escalação.

Processos tipo 2 - são processos de aplicação que podem interromper os processos tipo 3 e 4 e não podem ser interrompidos.

Processos tipo 3 - processos da aplicação que podem ser suspensos para a execução de processos tipo 1 e 2. São prioritários sobre os processos tipo 4 na escalação.

Processos tipo 4 - processos do sistema operacional que podem ser suspensos por processos tipo 1 e 2.

Dessa forma o sistema proporciona preempção. A prioridade do processo é fornecida na sua criação.

O processo é colocado na lista de processos ativos relativa à sua prioridade. A partir desse instante o processo estará ligado sempre à uma lista da sua prioridade. Quando o processo está em execução a sua prioridade é armazenada na variável PRIPROEX.

O tempo de execução permitido para os processos 1 e 2 é menor que o permitido para os processos 3 e 4. Isto se deve ao fato dos processos 3 e 4 poderem ser suspensos para a execução de processos prioritários. O núcleo possui o seu próprio WDT.

Para resolver o problema de acesso indevido a dados compartilhados por processos de prioridades diferentes de um mesmo programa, fica estabelecido que, no caso de processos do mesmo programa, os de menor prioridade não poderão ser suspensos para a execução dos processos de maior prioridade.

VI.2.2.3. Primitivas

As primitivas relacionadas abaixo pertencem ao núcleo e permitem a gerência dos processos sob seu controle e a utilização desses recursos pelo usuário.

STARTPR - Esta primitiva permite a um processo criar outro processo do mesmo programa, colocando-o no estado ativo. A sua execução só é aceita se o processo estiver parado e o número do programa do processo em execução for o mesmo do processo que está sendo ativado.

Entrada - Número do processo, número da instância, prioridade do processo, posição de execução e WDT.
 Saída - ok -> primitiva executada corretamente
 erro -> processo já criado
 não tem célula de processo livre
 parâmetros de entrada errados ou inconsistentes.

STOPROC - Esta primitiva permite ao processo que está executando provocar a sua própria parada.

Entrada - são obtidas na célula do processo em execução
 Saída - não há

ABORTPR - Permite a um processo parar outro processo do mesmo programa. Os processos tratadores de falha podem provocar a parada de outros processos do mesmo programa em caso de necessidade. O núcleo verifica a prioridade do processo em execução para reconhecer o processo que executou a chamada.

Entrada - número do processo, instância, prioridade do processo. Os demais dados são obtidos na célula do processo em execução.
 Saída - ok -> primitiva executada corretamente
 erro -> processo que chamou não pode abortar o outro processo. Processo não existe, parâmetros de entrada errados ou inconsistentes.

REATIPR - Esta primitiva permite a um processo se colocar ao final da lista de processos ativos relativos à sua prioridade, podendo dessa forma retornar à execução sem a necessidade de espera.

Entrada - posição a ser executada. As demais informações estão na célula do processo em execução.
 Saída - não há

ALTRAST - Permite ao usuário alterar a condição de rastreamento de um processo. Significa que os sinais enviados ou recebidos pelo processo em questão serão enviados para o terminal do operador caso o processo esteja em rastreamento.

Entrada - programa, processo, instância e condição de rastreamento.

Saída - OK ou ERRO (processo não existe)

ALTEPRI - Esta primitiva permite a um processo alterar a sua prioridade de execução. Os processos da aplicação podem passar de 2 para 3 ou de 3 para 2, enquanto que os processos do S.O. podem passar de 1 para 4 ou de 4 para 1. Ao terminar a sua execução o processo é colocado na lista relativa à sua nova prioridade.

Entrada - valor do WDT e nova prioridade

Saída - Ok ou erro (WDT ou prioridade não permitidas, erro de parâmetros de entrada).

A primitiva abaixo pertence ao configurador e é utilizada pelo núcleo durante a execução dessa função.

INICONF - Esta primitiva é chamada pelo núcleo após a iniciação do "hardware" e sua própria iniciação. Através dela o núcleo obtém o número de programas e o ponteiro do processo de iniciação de cada programa do processador local. Os primeiros ponteiros são dos processos de programas do S.O., seguindo-se os processos de programas do usuário. O núcleo executa os processos de iniciação um a um e a seguir entra em regime normal. Se não houver programas no processador local o núcleo chama periodicamente esta primitiva e nos intervalos executa testes de "HARDWARE". No caso da existência de proteção de memória fornece também os endereços limites de código e dados para cada programa.

Entrada - não há

Saída - número de programas. Um ponteiro para cada programa E limites de códigos e dados (caso exista proteção de memória).

A primitiva abaixo pertence ao S.O. e é utilizada pelo núcleo durante a execução dessa função.

INFPROC - Esta primitiva é utilizada pelo núcleo para avisar o sistema operacional todas as vezes em que um processo é ativado ou parado, permitindo ao sistema operacional manter controle sobre os programas do processador.

Entrada - número do programa, número do processo, número da instância e tipo de ação (ativo, parado).

Saída - não há

VI.2.2.4. Área de Comunicação

A passagem de dados nas chamadas de primitivas do núcleo é feita através de áreas ligadas a cada tipo de processo. Durante a execução o processo tem acesso somente à sua área de comunicação, à sua pilha, à sua área de dados e código, prevenindo dessa forma erros de manipulação de ponteiros pelo usuário. Através de PRIPROEX o núcleo identifica qual o buffer utilizado para a passagem de parâmetros e dados pelo processo em execução. As áreas são: BUFCOMO, BUFCOM1, BUFCOM2, BUFCOM3, BUFCOM4 e BUFNUC. As cinco primeiras são ligadas aos processos tipo 0, 1, 2, 3 e 4 respectivamente, enquanto a última é utilizada pelo núcleo para as suas chamadas ao sistema operacional ou configurador. O tipo e número de parâmetros e dados dependem da primitiva chamada. Desta forma é a primitiva que os identifica.

VI.2.2.5. Célula Descritora de Processo

Esta célula permite ao núcleo manter controle sobre os processos em execução no módulo local. Cada processo de cada programa e cada instância desse processo recebe um número obtido na fase de compilação por meio de uma biblioteca. Esta biblioteca possui cada programa, seus processos e suas instâncias e a cada novo programa, novo processo de um programa ou nova instância de um processo ele é colocado ao final de seus pares a fim de não alterar o número dos já existentes. Figura (VI.2).

Ao núcleo cabe apenas o controle sobre os processos e suas instâncias.

A célula é iniciada pelo núcleo quando um processo executa a primitiva STARTPR. É liberada quando o próprio processo executa a primitiva STOPPROC ou quando outro processo do mesmo programa executa a primitiva ABORTPR. Existe uma lista de células livres (LICEPROLIV) e várias listas de células utilizadas, que são:

lista de células de processos ativos de prioridade 1

(LICEPROAT 1)

lista de células de processos ativos de prioridade 2

(LICEPROAT 2)

lista de células de processos ativos de prioridade 3

(LICEPROAT 3)

lista de células de processos ativos de prioridade 4

(LICEPROAT 4)

lista de células de processos tratadores de falha ativos (prioridade 0) (LICEPROAT 0)

lista de células de processos 'a espera de sinal (prioridade 1)

(LICEPROSI 1)
 lista de células de processos 'a espera de sinal (prioridade 2)
 (LICEPROSI 2)
 lista de células de processos 'a espera de sinal (prioridade 3)
 (LICEPROSI 3)
 lista de células de processos 'a espera de sinal (prioridade 4)
 (LICEPROSI 4)

lista de células de processos de prioridade 1 'a espera de sinal ser recebido (LICEPRORE 1)
 lista de células de processos de prioridade 2 'a espera de sinal ser recebido (LICEPRORE 2)
 lista de células de processos de prioridade 3 'a espera de sinal ser recebido (LICEPRORE 3)
 lista de células de processos de prioridade 4 'a espera de sinal ser recebido (LICEPRORE 4)

lista de células de processos de prioridade 1 'a espera de temporização (LICEPROTE 1)
 lista de células de processos de prioridade 2 'a espera de temporização (LICEPROTE 2)
 lista de células de processos de prioridade 3 'a espera de temporização (LICEPROTE 3)
 lista de células de processos de prioridade 4 'a espera de temporização (LICEPROTE 4)

célula de processo em execução (CELPROEX)

célula de processo (CELPROSU) suspenso

célula de processo suspenso por erro (CELPROCERRO)

A variável PRIPROEX fornece a prioridade do processo em execução. Temos 5 tipos de processos. Processos do usuário que não podem ter a sua execução interrompida (prioridade 2), processos do usuário que podem ser interrompidos para a execução de processos de prioridade maior (prioridade 3). Os processos de prioridade 1 e 4 são do sistema operacional (vide OBS 1). Os processos 0 são processos tratadores de falha. O núcleo atualiza essa variável sempre que um novo processo é escalado e passa a ser o processo em execução. Se não há processo em execução o valor de PRIPROEX é 5. A prioridade de um processo é fornecida pela primitiva STARTPR, sendo que a partir deste momento o processo estará sempre ligado às listas de células desta prioridade.

Um processo pode ficar à espera de um sinal qualquer ou de um sinal específico.

Outra primitiva que atua sobre essa célula descritora de processo é a REATIPR. Esta primitiva coloca a célula do processo no final da lista de ativos referentes à sua prioridade.

A função do núcleo responsável pela gerência dessas células e' a função de controle de processos.

PONT. DE ENCADEAMENTO
Nº DO PROGRAMA
Nº DO PROCESSO
Nº DA INSTÂNCIA
VALOR DO WDT
RAST / NIUTRAT
PONT. DE ENCADEAMENTO P/ SINAIS DE PRIORIDADE \emptyset
PONT. DE ENCADEAMENTO P/ SINAIS DE PRIORIDADE 1
PONT. DE ENCADEAMENTO P/ SINAL ES. PERADO E POSIÇÃO A EXECUTAR
SINAL ESPERADO
POSIÇÃO A EXECUTAR

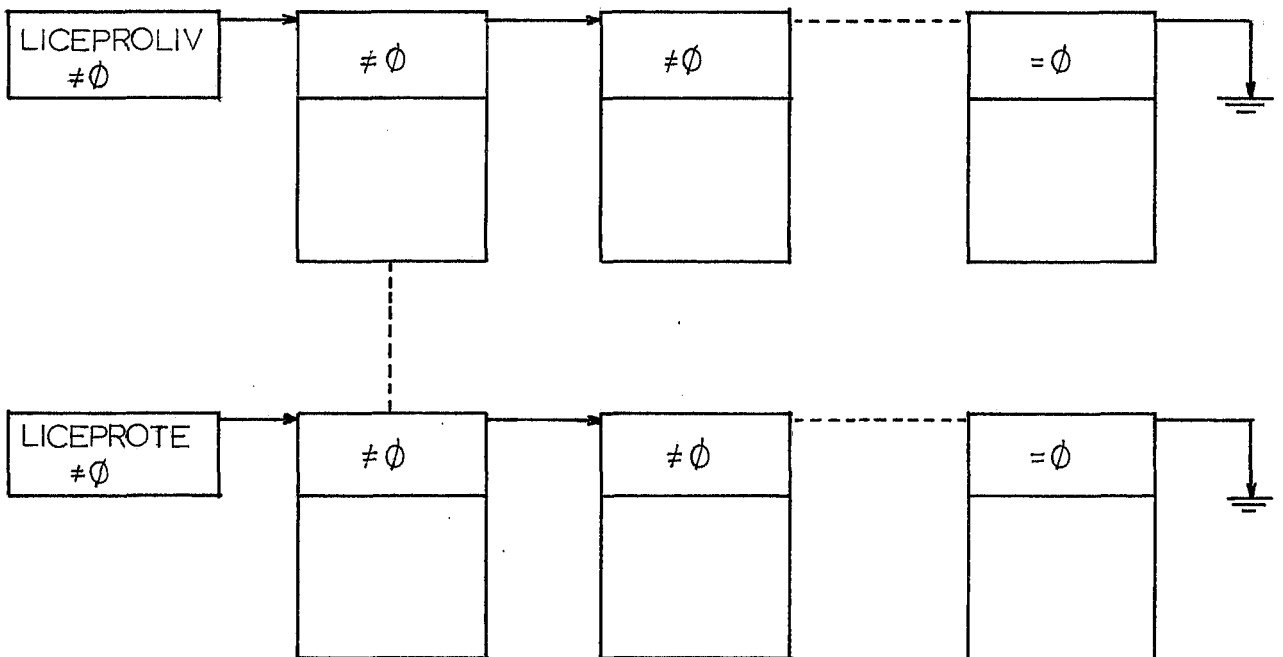


FIGURA (VI,2)

Ponteiro de encadeamento - fornece o ponteiro para a próxima célula da lista a qual pertence. Se o ponteiro é igual a zero ela é a última da lista

Número do programa - fornece a identidade do programa. Esse número pode ir de 0 a 255, sendo que cada programa recebe um único número dentro do sistema. Esse número é obtido a partir da compilação e é fornecido pela primitiva STARTPR.

Número do processo - identifica o processo dentro do programa. Cada processo de um programa recebe um número único de 0 a 256 fornecido através da primitiva STARTPR.

Número da instância - identifica a instância dentro do processo. Cada instância de um processo recebe um único número de 0 a 256. Fornecido através da primitiva STARTPR.

Valor do WDT - o WDT é o tempo de execução permitido para um processo executar. Este tempo evita que um processo tome tempo de processador acima de suas necessidades. Se um processo estourar o seu WDT ele é parado e a condição de erro é comunicada ao sistema operacional. Esse valor é fornecido pela primitiva STARTPR ou ALTEPRI. Pode ir de 0 a 256 vezes o valor do relógio de 4 ms.

Para processos do sistema operacional do tipo 4 varia de 100 a 256. Para processos do usuário do tipo 1 varia de 6 a 20. Para processos do usuário do tipo 2 varia de 21 a 60 e para processos do usuário do tipo 4 vai de 10 a 40. Para processos de iniciação é de 256 e para processos tipo 0 é de 6.

Rast/Nivtrat - o campo rastreamento indica se os sinais que se destinam ou são originados pelo processo devem ser visualizados pelo S.O.. O campo NIVTRAT indica se o usuário deseja tratar as falhas provocadas por esse processo. Alterado pelas primitivas ALTRAST e ALTENIV respectivamente.

Ponteiro de encadeamento para sinais de prioridade 0 recebidos pelo processo - este ponteiro fornece a posição do primeiro sinal da lista de sinais de prioridade 0 recebidos pelo processo. Se o valor do ponteiro é igual a zero não há sinal de prioridade 0 para o processo.

Ponteiro de encadeamento para sinais de prioridade 1 recebidos pelo processo - ídem caso anterior para sinais de prioridade 1.

Ponteiro de encadeamento para sinal esperado/posição a executar - fornece a posição do próximo sinal esperado pelo processo e sua correspondente posição de execução. O conjunto de informações constituído pelo ponteiro, sinal esperado e posição de execução é chamado de célula de execução condicional. Toda célula descritora de processo possui uma célula de execução condicional em seu próprio corpo. Se o ponteiro é igual a zero não há outras células de execução condicional.

Sinal esperado - fornece a identidade do sinal esperado pelo processo. Se seu valor for zero, o processo está a espera de qualquer sinal.

Posição a executar - fornece a posição do processo a executar no caso da chegada do sinal esperado.

São 50 células descritoras de processos. O núcleo varre as listas de células de processos sempre que há uma interrupção de relógio ou quando o controle do processador lhe é devolvido por um processo.

VI.2.2.6. Célula de Execução Condicional

Estas células fornecem a posição de execução de um processo no caso da chegada de um determinado sinal. Permite a um processo esperar vários sinais e para cada um deles executar procedimentos diferentes. Figura (VI.3).

Existe uma lista de células de execução condicional livres (LICEXCOLIV) e listas ocupadas ligadas a cada processo. Todo processo possui sempre uma célula de execução condicional embutida no corpo de sua célula descritora, sendo que ela é a primeira da lista de células de execução condicional do processo.

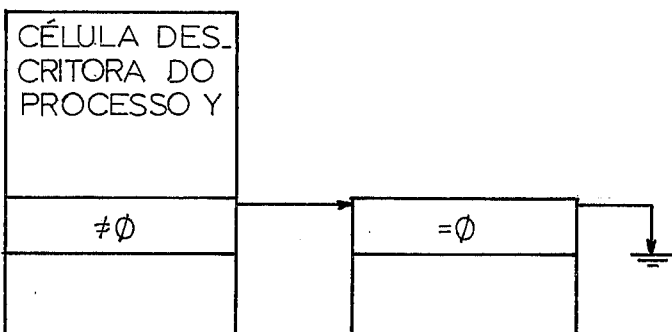
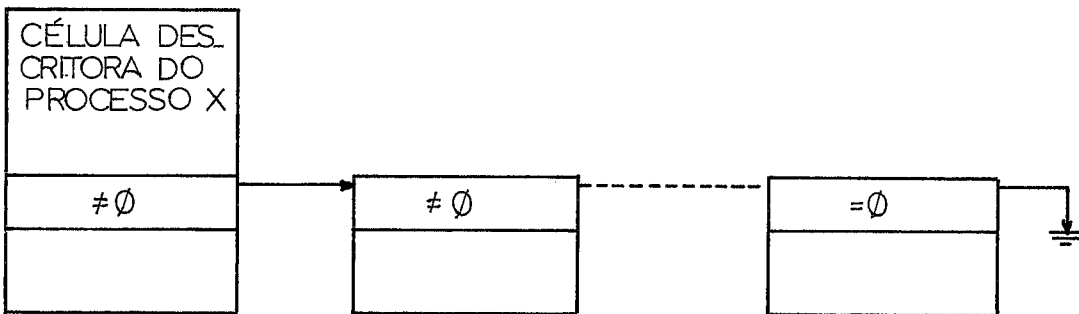
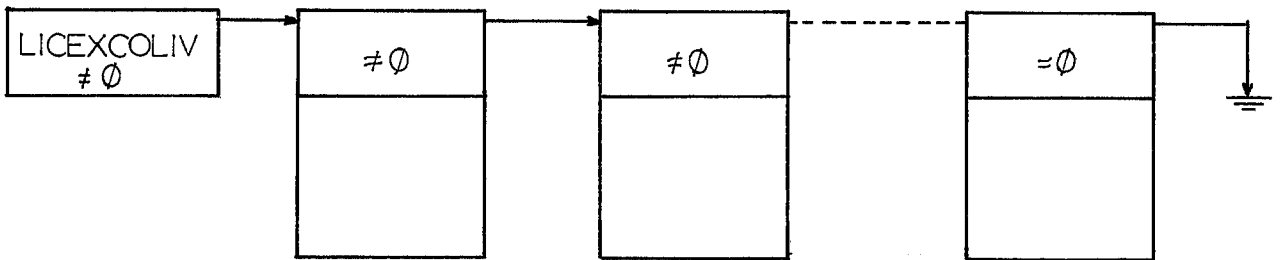
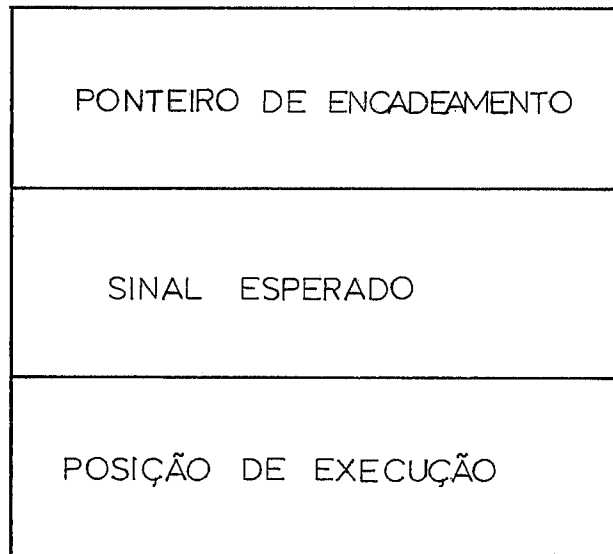


FIGURA (VI. 3)

Ponteiro de encadeamento - fornece a posição da próxima célula de execução condicional da lista. Se for igual a zero e' a última da lista.

Sinal esperado - fornece a identidade do sinal esperado pelo processo. Se for igual a zero o processo espera qualquer sinal.

Posição de execução - fornece a posição de execução do processo para o tratamento do sinal esperado.

São 50 células de execução condicional.

VI.2.2.7. Pilhas

O controle de processos e' responsável pelo controle e gerência das pilhas do sistema.

São 5 pilhas no sistema:

Pilha do núcleo (PIUSONUC)

Pilha de uso dos processos de prioridade 0 (PIUSOPRO 0)

Pilha de uso dos processos de prioridade 1 e 2 (PIUSOPRO 12)

Pilha de uso dos processos de prioridade 3 e 4 (PIUSOPRO 34)

Pilha de salvamento do contexto dos processos de prioridade 3 e 4 (PISALVPRO 34)

- Pilha do núcleo - Pilha utilizada pelo núcleo como àrea de trabalho. O núcleo mantém controle sobre essa àrea impedindo que a mesma seja ultrapassada. Se houver ultrapassagem aciona a falha para o sistema operacional e se reinicia.

- Pilha de uso dos processos de prioridade 0 - pilha utilizada como àrea de trabalho para esse tipo de processo.

- Pilha de uso dos processos de prioridade 1 e 2 - pilha utilizada como àrea de trabalho por esses processos.

- Pilha de uso dos processos de prioridade 3 e 4 - pilha utilizada como àrea de trabalho por esses processos.

- Pilha de salvamento do contexto dos processos de prioridades 3 e 4 - e' uma àrea utilizada para armazenar os registros utilizados por um processo quando o mesmo e' suspenso.

OBS. 1 :- Durante a sua execução os processos somente tem acesso à sua pilha de uso, à area de buffer para comunicação com o núcleo via primitivas, sua àrea de dados e código. No caso de tentativa de acesso fora dessas àreas ocorre vedação de memória.

OBS. 2 :- O núcleo ao receber o controle de um processo sempre restaura a sua pilha para o início.

OBS. 3 :- Um processo ao ser suspenso tem a sua célula colocada nessa condição. Antes de escalar processos tipo 3 e 4 o núcleo verifica essa célula.

VI.2.3. Controle de Temporização

As temporizações são gerenciadas pelo núcleo. A temporização pode ser utilizada para ativar um processo periodicamente a seu pedido ou a pedido de outro processo do mesmo programa, sendo que neste caso a temporização é cíclica. Um procedimento de um determinado processo também pode ser ativado uma única vez a cada pedido seu ou de outro processo do programa, configurando-se dessa forma uma temporização comum. Para prevenir bloqueio perpétuo existe a temporização de saída por tempo. Neste caso uma temporização é ativada a fim de que a mesma acorde o processo no caso da condição esperada não ser atendida no tempo de duração da temporização.

VI.2.3.1. Classes de Temporização

Uma temporização pode ser da classe rápida ou da classe lenta.

Classe rápida de temporização - possui um relógio base de 4 ms e quando a mesma estoura o seu tempo, o processo à ela ligado é retirado da fila de espera e colocado no início da fila de ativos relativos à sua prioridade. As temporizações deste tipo podem variar de 20 ms a 1 seg.

Classe lenta de temporização - possui um relógio base de 1 seg e quando a mesma termina o processo a ela ligado é colocado no fim da lista de processos ativos referentes à sua prioridade. As temporizações deste tipo podem variar de 1 seg a 4 m.

VI.2.3.2. Tipos de Temporização

Comum - o processo ligado a este tipo de temporização é ativado uma única vez; assim que o tempo termina.

Cíclica - o processo ligado a este tipo de temporização é ativado periodicamente, todas as vezes que a temporização vence. É renovada automaticamente.

Time-out - um processo ao pedir espera de um evento tem acionada uma temporização, pelo núcleo, para prevenir espera indefinida.

VI.2.3.3. Primitivas

As primitivas relacionadas abaixo pertencem ao núcleo e permitem ao usuário utilizar os recursos de temporização.

PEDTEMP - esta primitiva permite a um processo pedir ativações, periódicas ou não, dele mesmo após certo período. Permite também ao núcleo colocar condições de saída por tempo para os processos que pedem de um evento, evitando espera bloqueio perpétuo. Após a execução dessa primitiva o processo que estava executando é colocado na espera da temporização.

Entrada - número do processo, número da instância e prioridade do processo são obtidos na célula do processo em execução.

Classe de temporização - rápida ou lenta.

Tipo de temporização (periódica, comum, saída por tempo (time-out))

Tempo requisitado

Posição de execução

Saída - não ha' no caso de temporização de saída por tempo (time-out)

ok -> temporização aceita

erro -> não ha' célula livre

processo ja' tem outra temporização do tipo solicitado

erro nos parâmetros de entrada

OBS. Um processo so' pode ter uma temporização de cada tipo e classe por vez.

CANTEMP - esta primitiva permite a um processo cancelar uma temporização ja' ativada para ele ou para outro processo do mesmo programa.

Entrada - número do processo, número da instância, prioridade do processo (no caso de pedir cancelamento de temporização para outro processo)

Tipo de temporização

Classe de temporização

Temporização para o próprio processo ou não

Saída - ok -> temporização cancelada

erro -> não tem temporização

erro nos parâmetros de entrada

VI.2.3.4. Área de Comunicação

A passagem de dados e parâmetros nas chamadas das primitivas de temporização e' feita através das mesmas áreas definidas no controle de processos.

VI.2.3.5. Célula de Temporização

A célula de temporização Figura (VI.4) permite ao núcleo manter o controle sobre esse recurso do sistema. Existe uma lista de célula de temporização livre (LICETELIV) e listas de células de temporização ocupadas, que são:

- lista de células de temporização cíclica rápida (LICETECIRA)
- lista de células de temporização comum rápida (LICETECORA)
- lista de células de temporização de saída por tempo (time-out) rápida (LICETETIRA)
- lista de células de temporização cíclica lenta (LICETECILE)
- lista de células de temporização comum lenta (LICETECOLE)
- lista de células de temporização de saída por tempo (time-out) lenta (LICETETILE)

Duas primitivas agem sobre essas células. A primitiva PEDTEMP que aciona uma temporização, retirando uma célula da lista de livres e ocupando-a. A primitiva CANTEMP que cancela uma temporização, retirando uma célula de um dos tipos de temporização e colocando-a na lista de livres.

Ao vencer uma temporização da classe rápida, o processo a ela ligado e' colocado no início da lista de processos ativos relativa à sua prioridade. Se a temporização e' da classe lenta o processo a ela ligado e' colocado ao final da lista de processos ativos relativa à sua prioridade.

PONTEIRO DE ENCADEAMENTO
Nº DO PROGRAMA
Nº DO PROCESSO
Nº DA INSTÂNCIA
PRIORIDADE DO PROCESSO
VALOR DA TEMPORIZAÇÃO
VALOR ATUAL DA TEMPORIZAÇÃO
POSIÇÃO DE EXECUÇÃO (TIME-OUT)

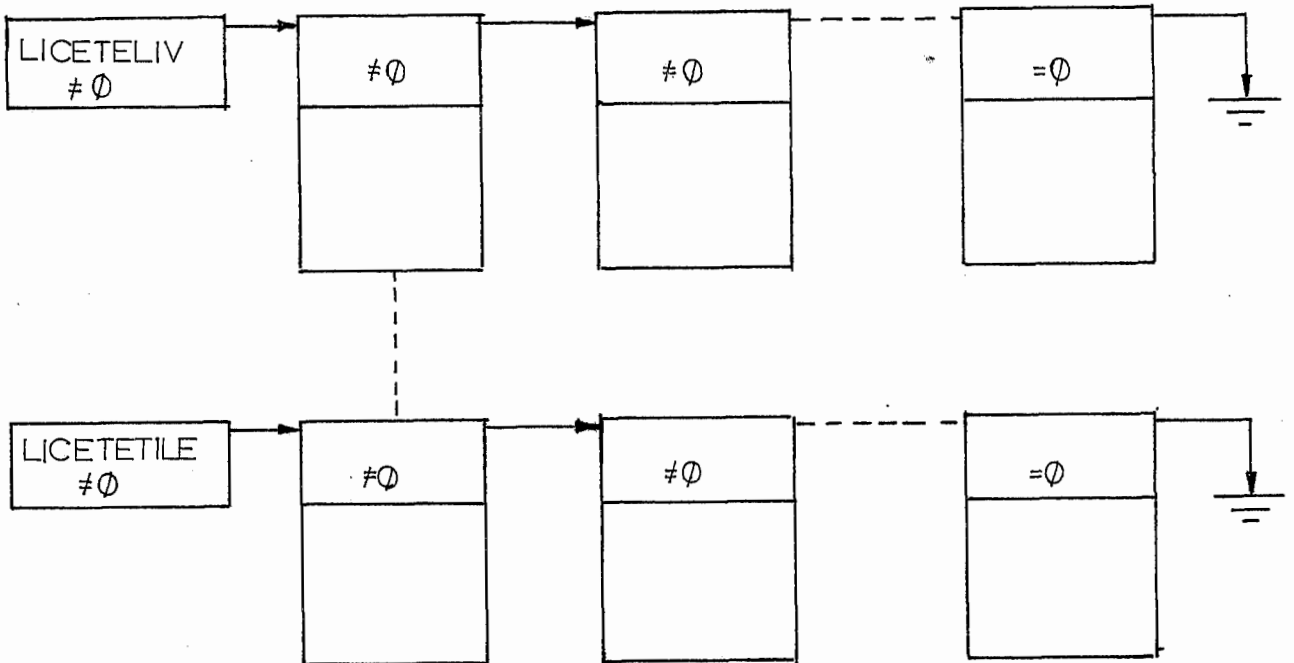


FIGURA (VI.4)

Ponteiro de encadeamento - fornece o ponteiro para a próxima célula da lista à qual pertence. Se o ponteiro é igual a zero ela é a última da lista.

Número do programa - indica o número do programa ao qual o processo que pediu a temporização pertence. Varia de 0 a 255.

Número do processo - indica o número do processo que pediu a temporização. Varia de 0 a 256.

Número da instância - fornece a instância do processo que pediu a temporização e que será ativada ao seu final. Varia de 0 a 256.

Prioridade do processo - contém a prioridade do processo que pediu a temporização. Permite ao núcleo obter a célula descritora do processo que ativou a temporização mais rapidamente.

Valor da temporização - informa ao núcleo o número de vezes que o relógio base da classe de temporização deve ocorrer antes de ativar o processo ligado à temporização. Pode variar de 1 a 255.

Valor atual da temporização - indica o momento em que o processo que ativou a temporização deve ser acordado. É iniciado com valor idêntico ao valor da temporização, sendo decrementado a cada ocorrência do relógio ligado à sua classe. Pode variar de 1 a 255 no início, sendo igual a zero na ativação do processo.

Posição de execução (time-out) - Indica para cada número onde o processo deve executar no caso de ocorrer um time-out.

São 32 células de temporização. A cada ocorrência de relógio base as células relativas à classe são variadas pelo núcleo para serem atualizadas. Se algum contador atingiu o valor zero, a função controle de temporização ativa a função controle de processos, passando-lhes os parâmetros da temporização ocorrida a fim de que o processo seja ativado. Após receber o controle de volta, termina a varredura e devolve o controle do processador a quem o detinha antes da ocorrência da interrupção do relógio.

VI.2.4. Controle de Sinais

Os sinais são gerenciados pelo núcleo através da função controle de sinais. Os sinais são utilizados pelos processos para comunicação e sincronismo. Quando um processo deseja fornecer informações a outro processo, se utiliza de um sinal com dados. No caso da sincronização, um processo se utiliza do sinal sem dados, para que uma determinada ação só ocorra após outra ação em outro processo.

VI.2.4.1. Classes de Sinais

Curto - o sinal curto não contém dados, servindo basicamente para sincronização entre processos.

Longo - o sinal longo possui área para armazenamento de dados, servindo dessa forma para a comunicação entre processos.

VI.2.4.2. Tipos de Sinal

O tipo de sinal está ligado à sua prioridade. A prioridade do sinal serve para permitir um não determinismo. Dessa forma, pode-se alocar urgência maior a determinados sinais, permitindo o seu tratamento antes de outros.

Prioridade 0 - são os sinais de maior prioridade.

Prioridade 1 - sinais de menor prioridade.

VI.2.4.3. Primitivas

As primitivas abaixo pertencem ao núcleo e permitem ao usuário utilizar o recurso de sinais do sistema.

SENDSIN - esta primitiva permite a um processo enviar uma mensagem a outro processo no mesmo processador ou em outro processador.

Entrada - Classe do sinal (curto ou longo) e tipo de transmissão pode ser:

Correio - Pode ser para sinais externos ou internos. O sinal é colocado em uma lista do correio, ficando nessa lista até ser recebido por outro processo que o peça. Nesse caso o sinal não tem destino específico e qualquer processo pode acessá-lo. Após ser acessado pelo primeiro processo, o sinal é liberado. O processo ao enviar esse sinal fica à espera de que algum processo o receba. A identidade do sinal também é fornecida pelo processo.

Difusão Interna - Para sinais externos ou internos. Neste caso o sinal é colocado na lista de Difusão Interna do processador, ligada a uma classe, conforme indicado no pedido (classe rápida ou lenta). Durante o período indicado no pedido, o sinal está disponível para todos os processos que o peça. Após o período, o sinal é liberado (lenta entre 1s e

4 min / rápida entre 20 ms e 1s). Neste caso o processo deve indicar se o sinal e' interno ou externo, sendo que neste caso deve fornecer o processador destino. Após enviar o sinal, o processo continua sua execução. A identidade do sinal também e' fornecida pelo processo.

Comum - Pode ser para sinais internos ou externos. Neste caso, o sinal e' enviado para um destino específico e deve ser colocado na lista de sinais de sua prioridade ligado ao processo. O processo deve fornecer o programa destino, processo destino, instância, identidade do sinal, prioridade do sinal e se o sinal e' externo ou interno. No caso de ser externo deve indicar se o processador destino ja' esta' determinado ou não. Após enviar o sinal, o processo continua sua execução normal.

Difusão Externa - Sómente para sinais externos. Pode ser conjugada com qualquer uma das anteriores. Neste caso o sinal e' enviado simultâneamente para todos os processadores do sistema.

Saída - OK

Erro -- Não ha' buffer de sinal

Parâmetros c/ erro ou inconsistentes

Módulo externo c/ defeito

RECESIN - esta primitiva permite a um processo requisitar um sinal específico, um sinal qualquer da lista do processo, um sinal de Correio, um sinal de difusão interna, ou um número determinado de sinais específicos. Pode também assinalar para o núcleo se deseja esperar no caso do sinal(is) pedido(s) não estar(em) disponível(is)

Entrada - Tipo de recepção

Tipo de recepção pode ser:

Correio - O sinal ou sinais são requisitados da lista do correio. Neste caso, o processo fornece o número de sinais e a identidade dos sinais. Os sinais devem ser fornecidos na sua prioridade de atendimento, sendo que neste caso, o primeiro a ser en-

contrado e entregue. Se não ha' sinal para o processo entre os fornecidos, o processo fica à espera desses sinais.

Difusão Interna - O sinal e' requisitado da lista de difusão interna. O processo deve fornecer a classe da lista (lenta/rápida), o número de sinais e a identidade deles (na ordem de prioridade para recepção) e se o processo deve ou não ser parado caso não haja sinal para ele.

Comum - O sinal e' requisitado da lista de sinais ligados ao processo. A 1a. lista a ser varrida tem prioridade 0. O processo deve fornecer o número de sinais, identidade dos sinais e se o processo deve ou não parar, caso não haja sinais para ele (dos pedidos).

Saída - OK (sinal é copiado no buffer)

(não tem sinal)

Erro (parâmetros de entrada errados ou inconsistentes).

OBS.1 :- No caso de espera, o processo deve fornecer também a posição de programa a ser executada no caso de vencimento de time-out.

OBS.2 :- O número máximo de sinais a ser recebido em cada pedido e' 6.

LIBESIN - esta primitiva permite a um processo liberar um sinal específico, alguns sinais específicos ou todos os sinais por ele enviados.

Entrada - identidade do sinal (= 0 todos os sinais)

Saída - sinais liberados (OK)
 não havia sinais (OK)
 parâmetros errados (erro)

SAVESIN - esta primitiva permite a um processo evitar que um sinal específico, alguns sinais específicos ou todos os sinais a ele destinados e não utilizados, sejam liberados após a sua execução. O SAVE só pode ser executado uma única vez sobre o mesmo sinal. Ao término da execução de um processo todos os sinais são liberados a exceção dos que tem SAVE pela primeira vez. Todos os sinais destinados a um processo, que e' parado, são liberados.

Entrada - identidade do sinal (= 0 qualquer sinal)

Saída - não havia sinal específico
 não havia qualquer sinal
 foi executado o SAVE

A primitiva abaixo pertence ao S.O. e é utilizada pelo núcleo durante a execução dessa função.

PRODES - primitiva do S.O. utilizada pelo núcleo para obter o processador destino quando o mesmo não está determinado.

Entrada - no. do programa

Saída - no. do processador ou não há processador c/
 programa.

VI.2.4.4. Área de Comunicação

A passagem de dados e parâmetros nas chamadas das primitivas de sinais é feita através das mesmas áreas definidas no controle de processos.

VI.2.4.5. Células de Sinais

As células de sinais são áreas de memória onde são colocadas as informações necessárias para o manuseio dos sinais pelo núcleo e processos do sistema. As células de sinais podem ser curtas e longas. As células curtas possuem somente o destino e origem do sinal, enquanto que as longas possuem uma área de dados.

Os sinais possuem duas prioridades: 0 e 1. A prioridade serve para determinar, no caso de haver sinais de diferentes prioridades para o processo, qual será entregue primeiro quando o processo pedir o recebimento de um sinal qualquer.

O processador destino de um sinal pode ser determinado pelo processo que o envia ou não determinado. No primeiro caso, o processo obtém quais processadores do sistema possuem o programa destino, através de uma primitiva do S.O., e escolhe a qual enviar. No segundo caso, o núcleo ao receber o pedido de envio de um sinal, aciona uma primitiva do S.O. que determina o primeiro processador que possui o programa destino e se é o processador local (o primeiro processador a ser varrido é o local). O núcleo ao receber o controle, coloca o sinal na fila externa ou na fila de sinais do processo.

O processo só pode enviar 10 sinais a cada vez que for escalado. O núcleo mantém controle sobre o número de sinais enviados por um processo. No caso de tentativa de envio de mais sinais que o permitido, o processo é parado e a falha é passada à função controle de falhas para armazenamento. Feito o controle o núcleo continua funcionando normalmente.

Se um sinal é enviado para um processo parado de um programa, o mesmo é liberado. No caso de envio de um sinal com destino inconsistente, o processo é parado e a falha é armazenada pelo controle de falhas. Após este controle o núcleo continua em operação normal.

Existem duas listas de células de sinais livres. (Uma lista de células curtas e outra de células longas). Existem ainda duas listas de células curtas e longas ocupadas, ligadas a cada processo do usuário (uma lista de sinais de prioridade 0 e outra de sinais de prioridade 1), uma lista de correio, uma lista de difusão interna e uma lista de sinais para envio externo.

Lista de células de sinais curtos livres	(LICESICULIV)
Lista de células de sinais longos livres	(LICESILOLIV)
Lista de correio	(LISICORREIO)
Lista de difusão interna rápida	(LISIDIFURA)
Lista de difusão interna lenta	(LISIDIFULE)
Lista de sinais p/ transmissão externa	(LISITRANEX)

a) Sinal curto. Figura (VI.5).

São 32 células de sinais curtos.

PONT. DE ENCADEAMENTO
Nº DE DADOS = 0
IDENTIDADE DO SINAL
DIFUSÃO INTERNA / CORREIO / PRIORIDADE DO SINAL
ENVRECEPT / RASTREAMENTO / SAVE
CONTADOR
PROCESSADOR DESTINO
PROGRAMA DESTINO
PROCESSO DESTINO
INSTANCIA DESTINO
PROCESSADOR ORIGEM
PROGRAMA ORIGEM
PROCESSO ORIGEM
INSTÂNCIA ORIGEM

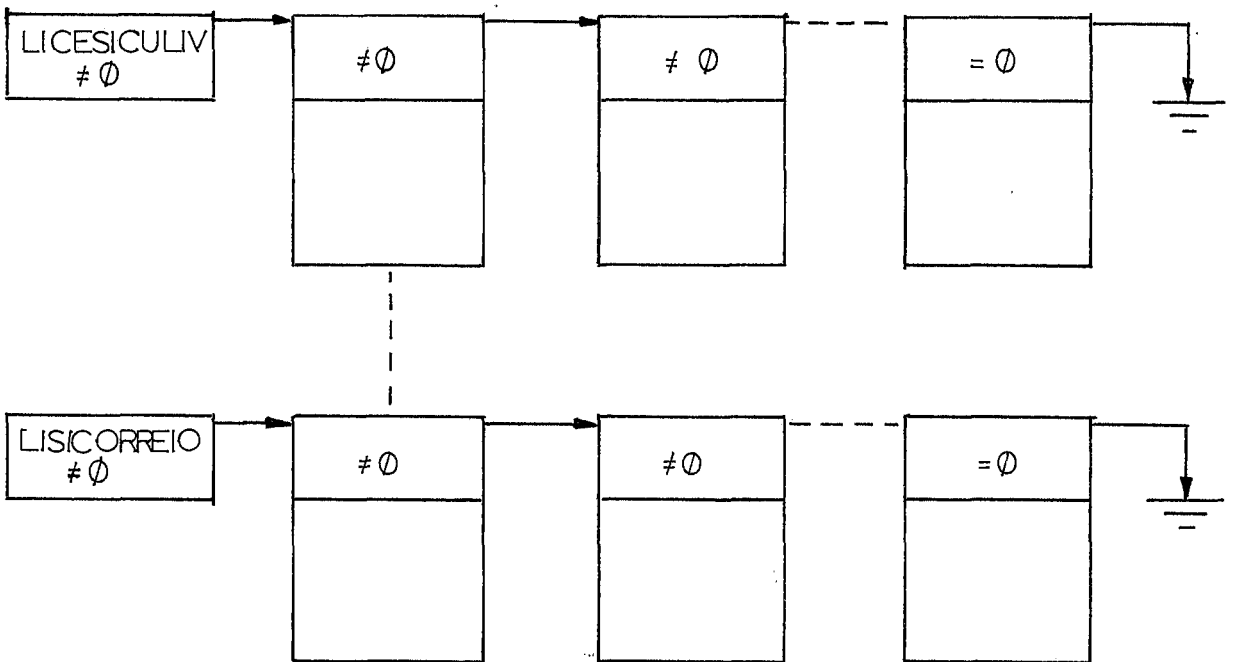


FIGURA (VI.5)

Ponteiro de encadeamento - fornece o ponteiro para a próxima célula da lista à qual pertence o sinal. Se o ponteiro é igual a zero ela é a última da lista.

Número de dados - no caso do sinal curto é sempre igual a 0.

Identidade do sinal - é o nome do sinal. Permite que um sinal seja diferenciado de outros sinais.

Se um sinal não tem nome, o valor desse campo é zero. Um processo pode pedir o recebimento de um ou mais sinais específicos. Existe uma biblioteca de sinais e cada sinal tem um nome único no sistema.

Difusão interna/Correio/Prioridade do sinal - O campo Correio indica que o sinal é enviado na qualidade de Correio e portanto, pode ser recebido pelo primeiro processo que o peça dentro do tempo de vida do Correio. A prioridade do sinal indica para os sinais que não são Correio ou Difusão interna, a lista na qual deve ser colocado junto ao processo. A lista de sinais com prioridade 0 é varrida antes da lista de sinais com prioridade 1. O campo de Difusão interna indica que um sinal pode ser recebido por todos os processos que o peçam dentro do período de tempo que ele permanecer na lista de Difusão interna.

Envrecept/Rastreamento/SAVE - O campo de rastreamento indica se um sinal deve ser enviado para o sistema operacional a fim de que o mesmo possa ser armazenado e posteriormente visualizado para o operador. Este recurso permite um acompanhamento do comportamento do sistema através das trocas de sinais. O campo SAVE permite a um processo retardar a liberação dos sinais a ele destinados até a sua próxima escalação. Desta forma os sinais destinados ao processo que está executando e que seriam liberados após a sua execução são guardados até a sua próxima escalação. O SAVE só é permitido uma única vez sobre o mesmo sinal. O campo Envrecept indica se o sinal foi enviado nesse modo.

Contador - Este campo fornece o tempo de permanência do sinal na lista de Difusão interna. É decrementado a cada relógio de lseg. ou 4 ms dependendo do tipo (lento ou rápido).

Processador destino - número do processador ao qual o sinal se destina. O número é colocado pelo núcleo após recebê-lo do sistema operacional. O sistema operacional devolve ao núcleo o número do processador destino e o número do processador origem, permitindo ao núcleo verificar se o sinal é interno ou externo.

Programa destino - identifica o programa para o qual o processo é destinado. Fornecido pelo usuário na chamada da primitiva.

Processo destino - identifica o processo do programa ao qual o sinal se destina. Fornecido pelo usuário na chamada da primitiva.

Instância destino - identifica a instância do processo destino a que se destina o sinal . Fornecido pelo usuário na chamada da primitiva.

Processador origem - indica o processador de onde o sinal foi enviado.

Programa origem - serve para identificar para o usuário que envia o sinal. É obtido pelo núcleo a partir da célula do processo em execução.

Processo origem - identifica o processo origem do programa . É obtido pelo núcleo a partir da célula do processo em execução.

Instância origem - identifica a instância do processo origem que envia o sinal . Obtida pelo núcleo a partir da célula do processo em execução.

b) Sinal longo. Figura (VI.6).

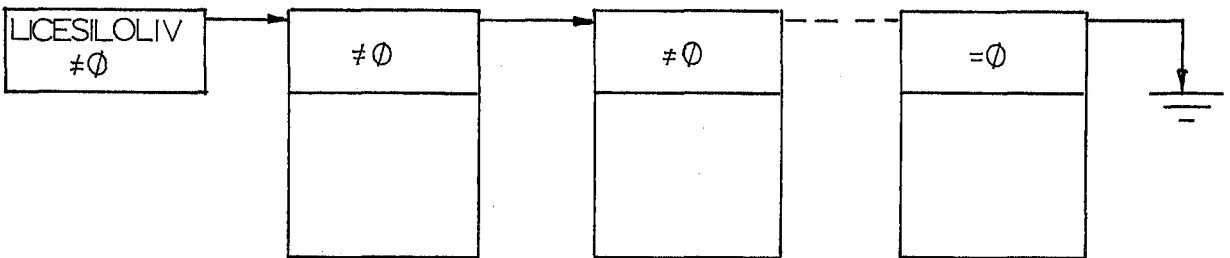
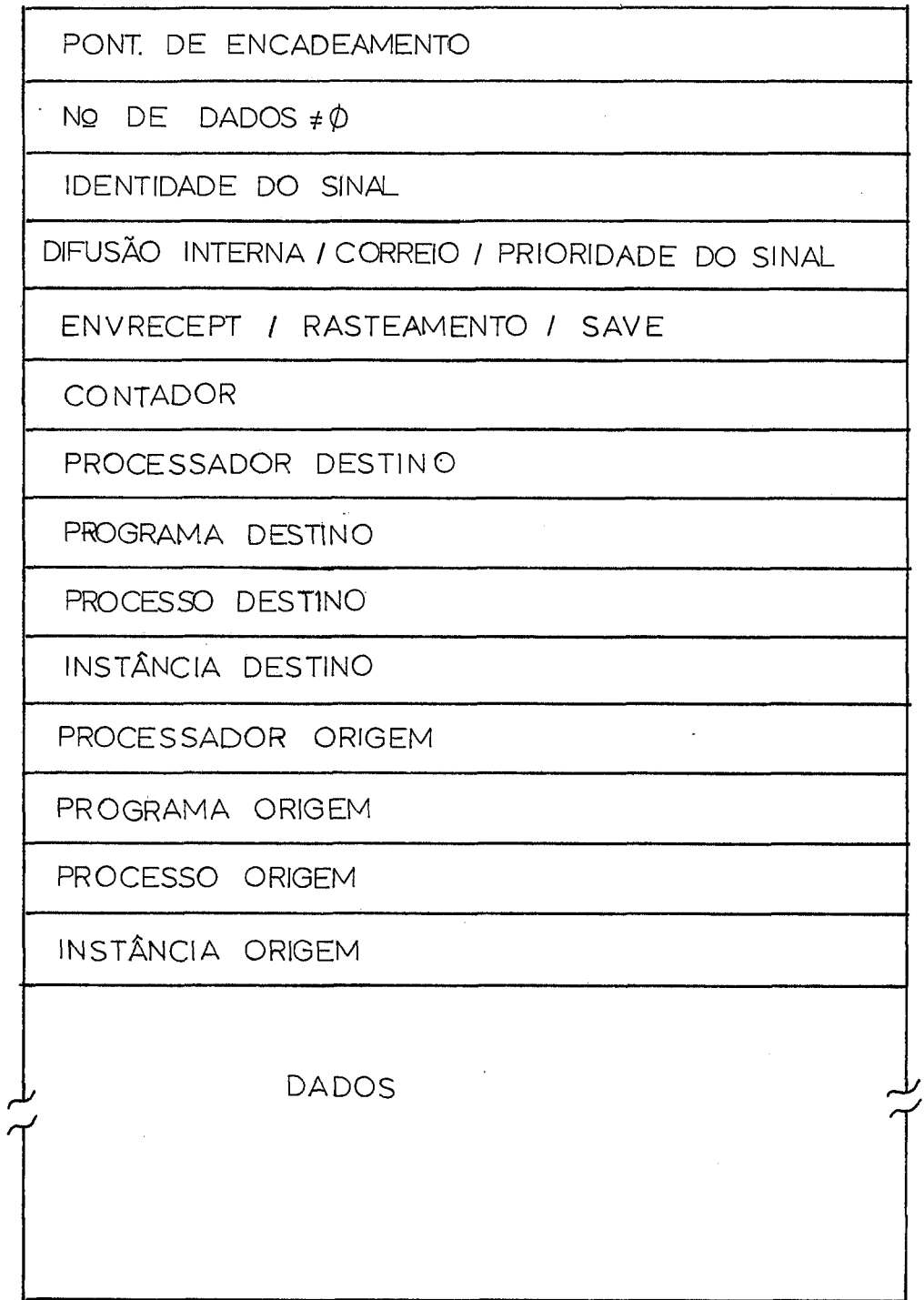


FIGURA (VI.6)

São 32 sinais.

As partes de controle do núcleo, destino e origem tem o mesmo significado do sinal curto.

Dados - e' uma `area onde os dados do usuário são colocados para serem transmitidos ao destino. No caso de transmissão externa são enviados somente os bytes de dados que o usuário realmente utiliza (evita gasto de tempo desnecessário na transmissão).

Número de bytes de dados - no caso de transmissão externa indica para o controle de comunicação externa o número de bytes de dados a ser transmitido.

VI.2.5. Controle de Comunicação Externa

Este controle e' responsável pela recepção e transmissão de sinais externos ao módulo local. Trabalha como um módulo escravo do controle de sinais.

a) Recepção

a recepção de sinais e' assíncrona ao funcionamento do módulo local e independente deste. O controle de comunicação externa possui um buffer do tamanho de um sinal longo independente da memória principal e responsável pelo armazenamento do sinal externo recebido. Na iniciação e após qualquer recepção o controle de comunicação externa se programa para uma recepção a pedido do controle de sinais. Após receber um sinal corretamente, o controle de comunicação externa coloca no registro de status a condição de recepção e aciona uma interrupção, inibindo outra recepção até que o sinal que chegou seja obtido pelo controle de sinais. O controle de sinais ao reconhecer a interrupção, verifica a condição de recepção. Se houve erro, passa a informação ao sistema operacional e reprograma o controle de comunicação externa. Se a recepção foi correta, o controle de sinais copia o buffer para um sinal interno do tipo e o coloca na lista relativa à ele, reprogramando a seguir, o controle de comunicação externa.

b) Transmissão

a transmissão de sinais e' feita a pedido do controle de sinais. Este ao verificar que existe um sinal na fila para transmissão externa, o coloca na condição de sinal a ser transmitido e pede ao controle de comunicação externa o seu envio. O controle de comunicação externa copia o sinal para o seu buffer (independente da memória principal) de transmissão e se prepara para a transmissão. O controle do processador e' devolvido ao controle de sinais e a transmissão se processa de forma independente. Ao final da transmissão o controle de comunicação externa coloca no registro de status de transmissão a condição da mesma. O controle de sinais antes de pedir a transmissão de novo sinal, verifica a condição de término do atual. No caso de falha, a condição e' passada ao sistema operacional e o sinal e' liberado. No caso de

transmissão ok, o sinal e' liberado.

c) Áreas de Trabalho

c.1) Buffer de transmissão - Área do tamanho de um sinal longo acessada por DMA. Utilizada para transmissão de sinais externos (BUFTRANEX).

c.2) Buffer de recepção - Área do tamanho de um sinal longo acessada por DMA. Utilizada para recepção de sinais externos (BUFRECEX).

c.3) Sinal de transmissão - variável que contém o ponteiro do sinal que está sendo transmitido externamente. No momento (POSITRANEX).

VI.2.6. Controle de Falhas

Esta função fornece os meios de tratamento das condições de falha detectadas pelo núcleo e a possibilidade do usuário implementar tratamento próprio para certos tipos de falha. É responsável também pelo acionamento do alarme de funcionamento do processador local. Necessária devido à aplicação (sistemas embutidos).

VI.2.6.1. Classes de Falhas

a) Classe 1

Estas falhas são sempre tratadas pelo núcleo. Dependendo do modo de tratamento podem provocar reiniciação automática do núcleo.

b) Classe 2

Estas falhas são sempre detectadas pelo núcleo. Não provocam reniciação do núcleo.

c) Classe 3

Estas falhas podem ser tratadas pelo núcleo ou pelo usuário dependendo do nível de tratamento estabelecido pelo processo em execução no momento. Se nada é informado ao núcleo este sempre trata as falhas no seu nível.

VI.2.6.2. Tipos de Falha

erro de memória ROM (classe 1)
 erro de memória RAM (classe 1)
 erro de UCP (classe 1)
 erro de WDT (classe 1)
 erro no circuito de proteção de memória (classe 1)
 erro no circuito de controle de pilha (classe 1)
 estouro de pilha de uso do núcleo (classe 1)
 violacao de proteção de memória pelo núcleo (classe 1)
 falha na comunicação externa (classe 2)
 estouro do WDT pelo núcleo (classe 1)
 falta de sinais (classe 2)
 sinal externo com destino errado (classe 2)
 estouro de pilha de uso do usuário (classe 3)
 estouro de WDT do usuário (classe 3)
 violacao de proteção de memória pelo usuário (classe 3)
 erro em parâmetros de primitivas (classe 3)
 tentativa de envio de mais sinais que o permitido (classe 3)
 chamada de primitiva especial por processo não autorizado
 (classe 3)

VI.2.6.3. Primitivas

As primitivas abaixo pertencem ao núcleo e permitem ao usuário a utilização dos recursos dessa função.

ALTENIV - Esta primitiva permite ao usuário estabelecer que certas falhas (fornecidas pela primitiva) serão tratadas pelo seu processo de tratamento de falhas. O núcleo verifica se o processo tratador de falhas existe. Somente as falhas de classe 3 podem ser tratadas pelo usuário.

Entrada - nível de tratamento:
 núcleo
 usuário

Saída - não ha' processo tratador
 OK

Esta primitiva altera o campo de nível de tratamento de falha na célula do processo.

PARAPRO - Esta primitiva permite ao processo de tratamento de falhas do programa parar o processador e transferir o controle para o depurador.

Entrada - não ha'

Saída - OK ou erro (não ha' depurador no processador)

INFALHA - Primitiva para fornecer os dados de uma falha detectada pelo núcleo. Ao usuário ou S.O. só pode ser chamada por processo tratador de falhas.

Entrada - Não ha'

Saída - Dados da falha ou erro (processo não pode ser executada a primitiva).

CELPROCERRO - Indica a célula do processo que provocou a falha. O processo desta célula esta' no estado erro.

A primitiva abaixo pertence ao S.O. e e' utilizada pelo núcleo durante o seu funcionamento.

INFALSO - Primitiva do S.O. usada pelo núcleo para informar suas falhas ao S.O.

Entrada - dados da falha

Saída - não ha'

As primitivas abaixo pertencem ao configurador e são utilizadas por essa função na sua execução.

INFMOD - Permite ao controle de falhas obter o modo de tratamento de falhas no processador local.

Entrada - não ha'

Saída - modo parado
modo contínuo

INFDEP - Esta primitiva informa a existência do depurador ao núcleo. Antes de transferir o controle ao depurador no caso de falha (quando o modo de tratamento indica parada) o núcleo verifica a existência ou não do depurador.

Entrada - não ha'

Saída - existe / não existe

VI.2.6.4. Processo de Tratamento de Falhas

Qualquer programa pode criar um processo para tratamento de falhas. Este processo é do tipo 0 e não pode ser suspenso durante a sua execução. So' esses processos podem executar a primitiva PARAPRO.

Existe uma lista de processos de tratamento de falhas ativos. Estes processos estão sempre ativos ou executando enquanto existirem.

VI.2.6.5. Nível de Tratamento

Podemos ter dois níveis de tratamento de falha:

Núcleo e usuário

a) Núcleo

As falhas classe 1 e 2 são sempre tratadas nesse nível.

As falhas de classe 3 podem ser tratadas nesse nível, dependendo da indicação de nível de tratamento na célula do processo em execução. Para as falhas classe 3 o núcleo identifica a falha, armazena os dados da falha em uma área de memória para esse fim, restabelece as condições para a continuação do funcionamento normal do sistema e prossegue operação normal.

Para a falha classe 2 o núcleo, após identificar a falha e armazenar os dados da mesma, verifica se o modo de tratamento de falhas do núcleo indica parada. Se indicar parada o núcleo entrega o controle ao depurador, caso contrário prossegue como nas falhas classe 3.

Para a falha classe 1 o núcleo após identificar a falha, armazena os dados da falha na área de memória e verifica o modo de tratamento. Se o modo de tratamento indicar parada o controle é entregue ao depurador, caso contrário o núcleo é reiniciado.

BUFALHA - Área de memória. Não é apagada pelo núcleo na iniciação nem é alterada pelo teste de RAM. Armazena os dados da falha detectada. Somente o controle de falhas pode escrever nessa área. Armazena os dados de uma falha. Os dados da falha incluem o contexto do processo em execução. Na primeira posição está o código da falha ocorrida.

MODTRAT - Indica no nível de tratamento do núcleo se o mesmo deve parar a sua execução ou não. Esta variável é obtida pelo controlador de falhas através da primitiva INFMOD na sua iniciação

(a primitiva e' atendida pelo configurador).

b) Usuário

Somente as falhas classe 3 podem ser tratadas nesse nível. Uma vez identificada a falha e' preenchida a área BUFALHA, o núcleo verifica o nível de tratamento na célula do processo em execução e, caso o usuário indique que deseja tratá-la, suspende o processo em execução por erro e escala o processo de tratamento de falhas do usuário. Ao receber o controle de volta, libera a célula do processo que estava suspenso por erro, todos os recursos por ele alocados e continua a execução normal. O processo de tratamento de falha pode provocar a parada do processador através da primitiva PARAPRO.

O sistema operacional ou o usuário pode obter os dados de uma falha, detectada pelo núcleo, através de uma chamada de primitiva. Após fornecer os dados ao S.O. ou usuário, o núcleo indica na primeira posição de BUFALHA que não existe falha.

VI.2.7. Controle de Interrupções

Este controle e' responsável pelo tratamento das interrupções que podem ocorrer no sistema.

Podemos ter seis tipos de interrupções:

- Interrupção por violação de memória
- Interrupção por estouro de WDT
- Interrupção de relógio de tempo real (4 ms)
- Interrupção de relógio de 1 segundo
- Interrupção de recepcao de sinal externo
- Interrupção por estouro de pilha
- Interrupção do debugger

a) Interrupção por violação de memória

Essa interrupção e' não mascarável e e' acionada sempre que um processo tenta acessar uma área de memória que não lhe pertence. O circuito de proteção de memória e' programado antes de se transferir o controle do processador ao processo. Uma vez ocorrida a falha o núcleo passa a informação ao sistema operacional e coloca o processo no estado parado. Necessita de um hardware adicional para o seu controle.

b) Interrupção por estouro de WDT

Essa interrupção é não mascarável, sendo acionada sempre que o tempo pedido por um processo ou pelo próprio núcleo for ultrapassado. Evita "loops" externos. O WDT é programado antes do controle ser transferido ao processo ou quando o núcleo recebe o controle do processador. No caso de estouro do WDT no processo, a falha é passada ao sistema operacional e o processo é parado. No caso de estouro do núcleo o mesmo é reiniciado e a informação de falha é passada ao sistema operacional. Se a falha persistir, o controle é passado ao depurador e o alarme é assinalado para o sistema operacional.

OBS. :- O núcleo possui um WDT no seu funcionamento normal e outro WDT na iniciação. Os processos de iniciação de cada programa também possuem um WDT default programado pelo núcleo.

c) Interrupção de relógio de tempo real (4 ms)

Esta interrupção é mascarável e ocorre a cada 4 ms. É utilizada para enviar sinais externos, atualizar dados de temporizações com tempo base de 4 ms e verificar escalação de processos. No caso de escalação de processos o núcleo verifica se existe um processo mais prioritário esperando escalação. Em caso positivo, transfere o controle para o processo prioritário e suspende o outro.

d) Interrupção de relógio de tempo base 1 segundo

Esta interrupção é mascarável e ocorre a cada 1 segundo, sendo utilizada para atualizar os dados de temporização de tempo base igual a 1 segundo.

e) Interrupção de recepção de sinal externo

Esta interrupção é mascarável e ocorre ao final da recepção de um sinal externo. É acionado pelo módulo de comunicação externa. Quando ocorre, o controle de sinais é acionado para o tratamento do sinal recebido e reprogramação do módulo de comunicação externa para nova recepção.

f) Interrupção por estouro de pilha

Esta interrupção é não mascarável, sendo acionada sempre que o usuário ou o núcleo ultrapasse o limite de sua pilha de uso. Necessita de um hardware adicional para o seu controle.

g) Interrupção do depurador

Esta interrupção é mascarável e é utilizada pelo programa depurador na implementação de suas funções de depuração.

CAPÍTULO VII

CONCLUSÃO

VII.1. RESULTADOS OBTIDOS

Os principais resultados obtidos neste trabalho são:

- a) O estudo e início da utilização da linguagem CHILL no Departamento de Comutação do CPqD. Este fato torna-se importante já que a linguagem CHILL foi escolhida como linguagem padrão para desenvolvimento de projetos no CPqD, devido à sua larga utilização como linguagem padrão no setor de telecomunicações.
- b) A especificação de um núcleo para sistemas de tempo real do tipo telefônico, que podera' servir como base para o desenvolvimento e implementação de um núcleo CHILL padrão para utilização nas centrais telefônicas da família TRÓPICO, atualmente em desenvolvimento no Departamento de Comutação do CPqD.
- c) A descrição de um Sistema Operacional, com características para utilização em sistemas do tipo telefônico, que podera' servir de ponto de partida para o desenvolvimento e implementação de um Sistema Operacional padrão para as centrais telefônicas anteriormente mencionadas.
- d) A reunião de dados bibliográficos sobre os assuntos mencionados na tese e que servirão como fonte de consultas para o grupo responsável pelo desenvolvimento de software básico para as centrais telefônicas da família TRÓPICO no CPqD.

VII.2. CONTINUAÇÃO DESTE TRABALHO

Como continuação deste trabalho podemos ter:

- a) O desenvolvimento e implementação de um Núcleo CHILL Padrão, a ser empregado nas centrais telefônicas da família TRÓPICO em desenvolvimento no Departamento de Comutação do CPqD. Esse núcleo tera' como processador hospedeiro o INTEL 286, escolhido como micro padrão para os projetos do CPqD.
- b) O desenvolvimento e implementação de um Sistema Operacional com características para emprego em sistemas do tipo telefônico. Esse Sistema Operacional Padrão sera' desenvolvido no Departamento de Comutação do CPqD para utilização nas centrais telefônicas da família TRÓPICO.

ANEXO IDESCRIÇÃO DO NÚCLEO EM LINGUAGEM DE ALTO NÍVEL

Neste anexo são apresentadas as várias funções do núcleo, descritas em uma linguagem de alto nível livre (PL/M LIKE). A base de dados e o corpo de cada função e seus algoritmos, assim como a estrutura global do núcleo, são apresentadas de modo a propiciar um entendimento geral do material apresentado nos capítulos anteriores.

Ao final deste anexo e' apresentado também um programa aplicativo na mesma linguagem como exemplo.

A utilização de uma linguagem livre facilita a descrição uma vez que não são obedecidas regras rígidas de sintaxe. O objetivo e' descrever as funções do núcleo e seus algoritmos de controle sem detalhes, a fim de possibilitar uma idéia do funcionamento do mesmo.

I.1. DESCRIÇÃO DO NÚCLEO

***** NUCLEO

**** BLOCO COMUM

*** AREA DE DEFINICAO

** DESCRICAO:

* ESTE NUCLEO IMPLEMENTA O AMBIENTE NECESSARIO
 * 'A EXECUCAO DOS ATRIBUTOS DE CONCORRENCIA
 * DA LINGUAGEM CHILL EM SISTEMAS DISTRIBUIDOS.
 * DESTINA-SE 'A APLICACAO EM SISTEMAS
 * TELEFONICOS (CPA-T).

** FUNCOES:

* CONTROLE DE INICIACAO
 * CONTROLE DE TESTES
 * GERENCIA DE PROCESSOS
 * CONTROLE DE TEMPORIZACAO
 * CONTROLE DE SINAIS
 * CONTROLE DE FALHAS

** CARACTERISTICAS:

* TAMANHO - X (CODIGO) E Y (DADOS)
 * LINGUAGEM - ASSEMBLER (8085) E PL/M
 * PROCESSADOR - INTEL 8085

** VERSAO:

* AUTORES - CARLOS CESAR FERREIRA ARAUJO
 * DATA - 10/12/1983
 * LOCAL - CPqD (TELEBRAS)
 * PROJETO - X

*** AREA DE COMUNICACAO

** BUFFERS DE COMUNICACAO

DCL BUFCOMO (80), BUFCOM1 (80), BUFCOM2 (80), BUFCOM3 (80),
 BUFCOM4 (80), BUFNUC (80) PUBLIC.

** PRIMITIVAS DO NUCLEO

DCL STARTPR, STOPPROC, ABORTPR, REATIPR, ALTERPRI, PEDTEMP,
 CANTEMP, SENDSIN, RECESIN, SAVESIN, LIBESIN, ALTENIV,
 PARAPRO, INFALHA, REINIPR PUBLIC.

** BLOCO DE COMUNICACAO GLOBAL

INCLUDE BLOCOGB

*** AREA DE DADOS

** CONSTANTES

.
 .
 .
 .
 .
 .

** VARIAVEIS (DATA SEGMENT)

.
 .
 .

.
.
.
** SUB-ROTINAS (CODE SEGMENT)

.
.
.
.
.
.
.
** MACROS

.
.
.
.
.
.
.
**** FIM DO BLOCO COMUM

```
**** BLOCO DAS FUNCOES
***  FUNCAO CONTROLE DE INICIACAO
**   AREA DE DEFINICAO

*   DESCRICAO:
*   * ESTA FUNCAO CONTROLA A INICIACAO DO NUCLEO.
*   CHAMADA POR:
*   * CONFIGURADOR, CONTROLE DE FALHAS, SISTEMA OPERACIONAL,
*   * DEPURADOR E USUARIO.
*   CHAMA:
*   * DEMAIS FUNCOES (TESTES, PROCESSOS, TEMPORIZACAO,
*   * SINAIS, FALHAS), DEPURADOR, CONFIGURADOR.
*   TAMANHO:
*   * X   (CODIGO) E   Y   (DADOS).
**   AREA DE COMUNICACAO
    .
    .
    .
**   AREA DE DADOS (DATA SEGMENT)
*   * CONSTANTES
    .
    .
    .
*   * VARIAVEIS (DATA SEGMENT)
    .
    .
    .
*   * SUB-ROTINAS (CODE SEGMENT)
    .
    .
    .
*   * MACROS
    .
    .
    .
```

** AREA DE COMANDOS.

INICIACAO:

BEGIN

INIBE INTERRUPTOES

INICIA PILHA DO NUCLEO

ZERA MEMORIA RAM

CALL INFDEP

; OBTEM INDICACAO DE EXISTENCIA
; DO DEPURADOR.

INICOO:

BUFNUC ^--- INDICA QUE INICIACAO CHAMA

CALL TEST

; TESTA TODO O HARDWARE BASICO

IF ERRO

THEN

BEGIN

INCREMENTA NUMERO DE TENTATIVAS

IF FIM

THEN

CALL FALHA

; CHAMA FALHA

ELSE

GO TO INICOO

END

ELSE

BEGIN

BUFNUC ^--- INDICA QUE E' INICIACAO

CALL FALHA

CALL PROCESSO

CALL TEMPORIZACAO

BUFNUC ^--- INDICA INICIACAO DA FUNCAO CONTROLE DE SINAIS

CALL SINAIS

INIC01:

BUFNUC ^--- INDICA INICIACAO DA COMUNICACAO EXTERNA

CALL SINAIS

IF ERRO

THEN

BEGIN

INCREMENTA NUMERO DE TENTATIVAS

IF FIM

THEN

CALL FALHA

ELSE

GO TO INIC01

END

ELSE

BEGIN

BUFNUC ^--- INDICA TRANSFERENCIA DE CONTROLE

GO TO PROCESSO

END

END.

```

* PRIMITIVAS DA FUNCAO CONTROLE DE INICIACAO:
REINIPR:
BEGIN
; FUNCAO - REINICIA PROCESSADOR
; ENTRADA - NAO HA'
; SAIDA - OK (PRIMITIVA EXECUTADA)
;          ERRO (CODIGO EM BUFALHA)
SALVA REGISTROS
IF PRIPROEX = 0 OU 5      ; QUEM CHAMOU PODE?
THEN
BEGIN
  IF MODTRAT INDICA PARADA
  THEN
  BEGIN
    BUFALHA ^--- INDICA FALHA NO USO DA PRIMITIVA PELO
                  PROCESSO EM EXECUCAO OU NUCLEO
    CALL FALHA
  END
  ELSE
    GO TO INICIACAO
  END
ELSE
BEGIN
  BUFALHA ^--- INDICA CHAMADA POR PROCESSO NAO AUTORIZADO
  CALL FALHA
  END
END
END          ; FIM DA PRIMITIVA REINIPR
END.
*** FIM DA FUNCAO CONTROLE DE INICIACAO

```

```
*** FUNCAO CONTROLE DE TESTES
** AREA DE DEFINICAO
* DESCRICAO:
* ESTA FUNCAO EXECUTA OS TESTES NO HARDWARE BASICO DO
* PROCESSADOR LOCAL (CPU, MEMORIA RAM, MEMORIA EPROM,
* RELOGIOS, CIRC. DE PROTECAO DE MEMORIA E CIRC. DE
* CONTROLE DAS PILHAS).
* CHAMADA POR:
* CONTROLE DE INICIACAO, GERENCIA DE PROCESSOS
* CHAMA:
* N.E.
* TAMANHO:
* X (CODIGO) E Y (DADOS)
** AREA DE COMUNICACAO
.
.
.
** AREA DE DADOS
* CONSTANTES
.
.
.
* VARIAVEIS (DATA SEGMENT)
.
.
.
* SUB-ROTINAS (CODE SEGMENT)
.
.
.
* MACROS
.
.
.
```


** AREA DE COMANDOS

TESTES:

BEGIN

IF INICIACAO CHAMA

THEN

BEGIN

INICIA NUMERO DO TESTE A REALIZAR

TEST00:

DO CASE NUMERO DO TESTE A REALIZAR

BEGIN

CALL TESTE DE CPU

CALL TESTE DE RAM

CALL TESTE DE EPROM

CALL TESTE DE RELOGIO

CALL TESTE DO CIRC. DE PROT. MEMORIA

CALL TESTE DO CIRC. DE CONT. DE PILHA

GO TO TEST01

END

IF FALHA

THEN

BEGIN

INDICA ERRO

BUFALHA ^--- DADOS DA FALHA OCORRIDA

RETORNA

END

PROXIMO TESTE

GO TO TEST00

TEST01:

INDICA QUE NAO HOUVE ERRO

INICIA NUMERO DO TESTE A REALIZAR

RETORNA

END

ELSE

BEGIN

DO CASE NUMERO DO TESTE A REALIZAR ; GERENCIA DE PROCESSOS CHAMOU

BEGIN

CALL TESTE DE CPU

CALL TESTE DE RAM

CALL TESTE DE EPROM

CALL TESTE DE RELOGIO

CALL TESTE DO CIRC. PROT. MEMORIA

CALL TESTE DO CIRC. CONT. DA PILHA

END

IF FALHA

THEN

INDICA ERRO

ELSE

INDICA TESTE OK

PROXIMO TESTE

IF FIM DOS TESTES

THEN

INICIA NUMERO DO TESTE A REALIZAR

RETORNA

END

END

*** FIM DA FUNCAO CONTROLE DE TESTES

*** FUNCAO GERENCIA DE PROCESSOS

** AREA DE DEFINICAO

* DESCRICAO:

* ESTA FUNCAO GERENCIA OS PROCESSOS, SUAS CELULAS E LISTAS

* E O USO DO PROCESSADOR PELOS MESMOS

* CHAMADA POR:

* CONTROLE DE INICIACAO, CONTROLE DE TEMPORIZACAO, CONTROLE

* DE SINAIS, INTERRUPCOES (4 ms, ESTOURO, WDT, ESTOURO DE

* PILHA, VIOLACAO DE MEMORIA), CONTROLE DE FALHAS

* CHAMA:

* CONTROLE DE TESTES, CONTROLE DE FALHAS, CONTROLE DE

* TEMPORIZACOES E CONTROLE DE SINAIS

* TAMANHO:

* X (CODIGO) E Y (DADOS)

** AREA DE COMUNICACAO

.

.

.

** AREA DE DADOS

* CONSTANTES

.

.

.

* VARIAVEIS (DATA SEGMENT)

.

.

.

* SUB-ROTINAS (CODE SEGMENT)

.

.

.

* MACROS

.

.

.

** AREA DE COMANDOS

PROCESSO:

BEGIN

IF INICIACAO CHAMA

THEN

BEGIN

ZERA AREA RAM

LIBERA CELULAS DE PROCESSOS

INICIA PILHA DE USO DOS PROCESSOS 0, 1, 2, 3 e 4

INICIA PILHA DE CONTEXTO DOS PROCESSOS 3 e 4

COLOCA DEMAIS LISTAS COMO VAZIAS

PRIPROEX ^--- 5 (NUCLEO EM EXECUCAO)

RETORNA

END

ELSE

OBTEMPROCINIC: ; NAO E' INIACAO

BEGIN

BUFNUC ^--- NUMERO DE PROGRAMAS = 0

CALL INICONF ; OBTEM NUMERO DE PROGRAMAS E

; PROCESSOS DE INICIACAO DE CADA

; PROGRAMA

DO WHILE NUMERO DE PROGRAMAS = 0

CALL TESTE ; TESTA HARDWARE BASICO

IF ERRO

THEN

CALL FALHA

CALL INICONF

END

; ESCALA PROCESSOS DE INICIACAO

ESCALAPROCINIC:

DO WHILE EXISTE PROCESSO DE INICIACAO

OBTEM PROCESSO DE INICIACAO ATUAL

DECREMENTA NUMERO DE PROGRAMAS

ESCALA PROCESSO DE INICIACAO

END

VARRER: ; REGIME NORMAL

BEGIN ; VARRE LISTAS DE PROCESSOS

IF EXISTE PROCESSO 1 ATIVO

THEN

BEGIN

OBTEM PRIMEIRA CELULA DA LISTA DE PROCESSOS 1 ATIVOS

REORDENA LISTA

CALL ESCALA

END

IF EXISTE PROCESSO 2 ATIVO

THEN

BEGIN

OBTEM PRIMEIRA CELULA DA LISTA DE PROCESSOS 2 ATIVOS

REORDENA LISTA

CALL ESCALA

END

IF EXISTE PROCESSO SUSPENSO

THEN

BEGIN

OBTEM CELULA DO PROCESSO EM CELPROSU

INDICA QUE NAO TEM PROCESSO SUSPENSO

```
        CALL ESCALA
END
IF EXISTE PROCESSO 3 ATIVO
THEN
BEGIN
    OBTEM PRIMEIRA CELULA DA LISTA DE PROCESSOS 3 ATIVOS
    REORDENA LISTA
    CALL ESCALA
END
IF EXISTE PROCESSO 4 ATIVO
THEN
BEGIN
    OBTEM CELULA DA LISTA DE PROCESSOS 4 ATIVOS
    REORDENA LISTA
    CALL ESCALA
END
                                ; NAO TEM PROCESSO ATIVO
CALL TESTE
IF ERRO
THEN
    CALL FALHA
GO TO VARRER
END
ESCALA:
BEGIN
COLOCA PRIORIDADE DO PROCESSO EM PRIPROEX
COLOCA CELULA DO PROCESSO COMO CELULA EM EXECUCAO
IF EXISTE CIRCUITO DE WDT
THEN
    PROGRAMA WDT COM VALOR DA CELULA
IF EXISTE CIRCUITO DE PROTECAO DE MEMORIA
THEN
    OBTEM LIMITES DE ACESSO E PROGRAMA CIRCUITO
    DE PROTECAO DE MEMORIA
IF EXISTE CIRCUITO DE CONTROLE DE PILHA
    PROGRAMA CIRCUITO COM LIMITE DA PILHA REFERENTE
    AO TIPO DO PROCESSO
INICIA PILHA DO PROCESSO
DISPARA WDT
TRANSFERE CONTROLE PARA POSICAO INDICADA NA CELULA DO
PROCESSO
END
END
```

* PRIMITIVAS DA FUNCAO GERENCIA DE PROCESSOS

```

STARTPR:
BEGIN
  ; FUNCAO - ATIVAR PROCESSO
  ; ENTRADA - No. DO PROGRAMA, No. DO PROCESSO, INSTANCIA,
  ;           PRIORIDADE, WDT E POSICAO DE EXECUCAO.
  ; SAIDA - OK (PRIMITIVA EXECUTADA)
  ;           ERRO (PARAMETRO COM ERRO, NAO TEM CELULA LIVRE)
  ;           DO CASE PRIORIDADE DO PROCESSO ; (PRIPROEX = PRIO-
  ;                                           ; RIDADE DO PROCES-
  ;                                           ; SO EM EXECUCAO)

  BUFCOMO
  BUFCOM1
  BUFCOM2
  BUFCOM3
  BUFCOM4

END
IF NUMERO DO PROGRAMA EM EXECUCAO = NUMERO DO PROGRAMA
DO PROCESSO ATIVADO
THEN
BEGIN
  IF WDT DENTRO DOS LIMITES PARA TIPO DO PROCESSO
  THEN
  BEGIN
    IF DEMAIS PARAMETROS OK
    THEN
    BEGIN
      OBTEM PRIMEIRA CELULA LIVRE
      PREENCHE COM DADOS DO PROCESSO
      REORDENA LISTA DE CELULAS LIVRES
      COLOCA CELULA DO PROCESSO NO FINAL DA
      LISTA DE CELULAS DE PROCESSOS ATIVOS DO
      SEU TIPO.
      INDICA SUCESSO NA CHAMADA DA PRIMITIVA NO
      BUFFER DE COMUNICACAO RELATIVO AO PROCESSO
      EM EXECUCAO
      CALL INPROC ; INFORMA AO S.O. QUE PROCESSO
                  ; FOI ATIVADO

      RETORNA
    END
  END
END
ELSE ; ERRO NA CHAMADA
BEGIN
  BUFALHA ^--- CODIGO DO ERRO ; PARAMETROS COM ERRO OU
                E DADOS ; FALTA DE CELULA LIVRE
  IF PARAMETROS ERRADOS
  THEN
    SUSPENDE POR ERRO PROCESSO QUE CHAMOU
  CALL FALHA
  PARA O PROCESSO SUSPENSO POR ERRO
  RETORNA
END
END ; FIM DA PRIMITIVA STARTPR
STOPPROC:
BEGIN
  ; FUNCAO = PARAR PROCESSO EM EXECUCAO

```

```

; ENTRADA - CELULA DO PROCESSO EM EXECUCAO
; SAIDA - NAO HA'
  OBTEM CELULA DO PROCESSO EM EXECUCAO
  LIBERA CEL. DE SINAIS E CEL. DE EXECUCAO CONDICIONAL DO
  PROCESSO
  COLOCA CELULA NO FINAL DA LISTA DE CELULAS LIVRES
  INDICA QUE NAO EXISTE PROCESSO EM EXECUCAO
  CALL INFPROC ; INFORMA PROCESSO PARADO
  PULA PARA VARRER
END ; FIM DA PRIMITIVA STOPROC

ABORTPR:
BEGIN
; FUNCAO - PARAR PROCESSO
; ENTRADA - NUMERO DO PROGRAMA, NUMERO DO PROCESSO, NUMERO DA
; INSTANCIA, PRIORIDADE DO PROCESSO
; SAIDAS - OK (PRIMITIVA EXECUTADA)
; ERRO (PARAMETROS DE ENTRADA ERRADOS. EX: PROCESSO NAO
; EXISTE
; PROCESSO QUE CHAMOU NAO PODE EXECUTAR A PRIMITIVA)
  IF PRIPROEX = 0
  THEN
  BEGIN
    VARRE LISTA DE PROCESSOS ATIVOS COM PRIORIDADE FORNECIDA
    IF NAO ENCONTROU
    THEN
    BEGIN
      VARRE LISTA DE PROCESSOS EM ESPERA COM PRIORIDADE
      FORNECIDA
      IF NAO ENCONTROU
      THEN
      BEGIN
        VERIFICA PROCESSO SUSPENSO
        IF NAO ENCONTROU
        THEN
        BEGIN
          VERIFICA PROCESSO SUSPENSO POR ERRO
          IF NAO ENCONTROU
          THEN
          BEGIN
            BUFALHA ^--- CODIGO E DADOS DO ERRO
            (PROCESSO NAO EXISTE)

            CALL FALHA
            RETORNA
          END
        END
      END
    END
  END
ELSE
BEGIN ; ENCONTROU PROCESSO
  OBTEM CELULA DO PROCESSO DA LISTA EM QUESTAO
  LIBERA CELULAS DE SINAIS, CELULAS DE EXECUCAO
  CONDICIONAL DO PROCESSO E CELULAS DE TEMPORIZACAO
  (SE TIVER)
  COLOCA CELULA NO FINAL DA LISTA DE CELULAS LIVRES

```

```

                REORDENA LISTA EM QUESTAO
                INDICA EXECUCAO OK
                CALL INFPROC          ; INFORMA PROCESSO PARADO AO
                RETORNA              ; S.O.
        END
    END
    ELSE
    BEGIN
                ; PROCESSO NAO PODE CHAMAR A PRIMITIVA
        BUFALHA ^--- CODIGO E DADOS DO ERRO
        CALL FALHA
        RETORNA
    END
END          ; FIM DA PRIMITIVA ABORTPR

REATIPR:
BEGIN
; FUNCAO - REATIVAR PROCESSO EM EXECUCAO
; ENTRADA - POSICAO A SER EXECUTADA, WDT (SE HOVER MUDANCA);
;          - DEMAIS DADOS NA CELULA DO PROCESSO EM EXECUCAO
; SAIDA - NAO HA'
        OBTEM CELULA DO PROCESSO EM EXECUCAO
        ATUALIZA CELULA COM DADOS DE ENTRADA (POSICAO DE EXECUCAO E
        WDT)
        COLOCA NO FINAL DA LISTA DE CELULAS DE PROCESSOS
        ATIVOS RELATIVOS A SUA PRIORIDADE
        INDICA QUE NAO EXISTE PROCESSO EM EXECUCAO
        PULA PARA VARRER
END          ; FIM DA PRIMITIVA REATIPR

ALTEPRI:
BEGIN
; FUNCAO - ALTERA PRIORIDADE DO PROCESSO EM EXECUCAO*
; ENTRADA - NOVA PRIORIDADE E WDT
; SAIDA - OK (PRIMITIVA EXECUTADA)
;          - ERRO (PROCESSO NAO PODE TER PRIORIDADE PEDIDA)
        IF PRIPROEX = 1 E PRIORIDADE PEDIDA = 4 OU PRIPROEX = 4 E
        PRIORIDADE PEDIDA = 1 OU PRIPROEX = 2 E PRIORIDADE PEDIDA
        = 3 OU PRIPROEX = 3 E PRIORIDADE PEDIDA = 2
        THEN
        BEGIN
                IF WDT DENTRO DOS LIMITES
                THEN
                BEGIN
                        PRIPROEX = NOVA PRIORIDADE
                        COLOCA NOVO WDT NA CELULA
                        INDICA SUCESSO NA CHAMADA DA PRIMITIVA
                        RETORNA
                END
                ELSE
                BEGIN
                        BUFALHA ^--- CODIGO E DADOS DO ERRO (PARAMETRO WDT
                        ERRADOS NA CHAMADA DE ALTEPRI)
                        SUSPENDE PROCESSO EM EXECUCAO POR ERRO
                        INDICA QUE NAO EXISTE PROCESSO EM EXECUCAO
                        CALL FALHA
                END
        END

```



```

                PARA PROCESSO SUSPENSO POR ERRO
                PULA PARA VARRER
            END
        END
    ELSE
    BEGIN
                ; PROCESSO NAO PODE TER PRIORIDADE
                ; PEDIDA
        BUFALHA ~--- CODIGO E DADOS DO ERRO (PRIORIDADE NA
                CHAMADA DE ALTEPRI)
        SUSPENDE PROCESSO EM EXECUCAO POR ERRO
        INDICA QUE NAO EXISTE PROCESSO EM EXECUCAO
        CALL FALHA
        PARA PROCESSO SUSPENSO POR ERRO
        PULA PARA VARRER
    END
END
                ; FIM DA PRIMITIVA ALTEPRI

ALTRAST:
BEGIN
; FUNCAO - ALTERA CONDICAO DE RASTREAMENTO DO PROCESSO
; ENTRADA - NUMERO DO PROGRAMA, NUMERO DO PROCESSO, NUMERO DA
                INSTANCIA E CONDICAO DE RASTREAMENTO
; SAIDA - OK
                ERRO (PROCESSO NAO EXISTE)
        OBTEM DADOS DE ENTRADA NO BUFFER DE COMUNICACAO
        RELATIVO AO PROCESSO EM EXECUCAO
        VARRE LISTAS DE PROCESSOS
        IF ENCONTROU PROCESSO
        THEN
        BEGIN
                ALTERA NA CELULA CONDICAO DE RASTREAMENTO
                INDICA SUCESSO NA CHAMADA DA PRIMITIVA
                RETORNA
        END
        ELSE
        BEGIN
                BUFALHA ~--- CODIGO ERRADOS DO ERRO (PROCESSO NAO
                EXISTE)
                SUSPENDE PROCESSO POR ERRO
                CALL FALHA
                PARA PROCESSO SUSPENSO POR ERRO
                RETORNA
        END
END
                ; FIM DA PRIMITIVA ALTRAST

*** FIM DA FUNCAO GERENCIA DE PROCESSOS

```

```
*** FUNCAO CONTROLE DE TEMPORIZACOES
** AREA DE DEFINICAO
* DESCRICAO:
  * ESTA FUNCAO FORNECE OS RECURSOS DE TEMPO AOS PROCESSOS
* CHAMADA POR:
  * CONTROLE DE INICIACAO, GERENCIA DE PROCESSOS,
  * INTERRUPCOES DE 1 s e 4 ms
* CHAMA:
  * CONTROLE DE FALHAS, GERENCIA DE PROCESSOS
* TAMANHO:
  * X (CODIGO) E Y (DADOS)
** AREA DE COMUNICACAO
.
.
.
** AREA DE DADOS
* CONSTANTES
.
.
.
* VARIAVEIS (DATA SEGMENT)
.
.
.
* SUB-ROTINAS (CODE SEGMENT)
.
.
.
* MACROS
.
.
.
```

** AREA DE COMANDOS

TEMPORIZACAO:

BEGIN

IF INICIACAO

THEN

BEGIN

LIBERA CELULAS DE TEMPORIZACAO COLOCANDO-AS NA LISTA DE
CELULAS LIVRES

INDICA COMO VAZIAS TODAS AS DEMAIS LISTAS

RETORNA

END

IF SINAIS

THEN

BEGIN

; TIME-OUT

IF EXISTE CELULA DE TEMPORIZACAO LIVRE

THEN

BEGIN

ALOCA UMA CELULA

PREENCHE COM DADOS FORNECIDOS EM BUFNUC

COLOCA CELULA NA LISTA DE TEMPORIZACAO DE TIME OUT

DA CLASSE INDICADA NO PEDIDO

RETORNA

END

ELSE

BEGIN

BUFALHA ~--> CODIGO E DADOS DO ERRO (FALHA DE CELULA DE
TEMPORIZACAO LIVRE)

CALL FALHA

RETORNA

END

END

IF PROCESSO

THEN

BEGIN

LIBERA CELULA DE TEMPORIZACAO DO PROCESSO INDICADO

RETORNA

END

* PRIMITIVAS DA FUNCAO CONTROLE DE TEMPORIZACOES

PEDTEMP:

BEGIN

; FUNCAO - LIGAR UMA TEMPORIZACAO

; ENTRADA - NUMERO DO PROGRAMA, NUMERO DO PROCESSO, NUMERO DA
; INSTANCIA, PRIORIDADE DO PROCESSO, CLASSE DA
; TEMPORIZACAO, TIPO DA TEMPORIZACAO, VALOR.

; SAIDA - OK

; - ERRO (NAO HA' CELULA LIVRE, PROCESSO JA' TEM TEMPO-
; RIZACAO DO MESMO TIPO PENDENTE OU ERRO NOS PARAMETROS
; DE ENTRADA)

IF PARAMETROS OK (PRIORIDADE, CLASSE E TIPO)

THEN

BEGIN

IF EXISTE CELULA LIVRE

THEN

BEGIN

OBTEM CELULA LIVRE

PREENCHE COM DADOS FORNECIDOS CELULA DE TEMPORIZACAO

COLOCA POSICAO DE EXECUCAO NA CELULA DO PROCESSO

EM EXECUCAO

COLOCA CELULA DE TEMPORIZACAO NA LISTA RELATIVA AO

SEU TIPO E CLASSE

END

END

ELSE

BEGIN

BUFALHA ^--- CODIGO E DADOS DO ERRO (NAO HA' CELULA
LIVRE, ERRO NOS PARAMETROS DE ENTRADAS)

CALL FALHA

RETORNA

END

END

; FIM DA PRIMITIVA PEDTEMP

CANTEMP:

BEGIN

; FUNCAO - CANCELA TEMPORIZACAO

; ENTRADA - NUMERO DO PROGRAMA, NUMERO DO PROCESSO, NUMERO DA
; INSTANCIA, PRIORIDADE DO PROCESSO, TIPO E CLASSE DA
; TEMPORIZACAO

; SAIDA - OK

; ERRO (PROCESSO NAO TINHA TEMPORIZACAO, NUMERO DO
; PROGRAMA FORNECIDO PROGRAMA EM EXECUCAO, ERRO
; NOS DADOS DE TIPO, CLASSE OU PRIORIDADE)

IF PARAMETROS OK (NUMERO DO PROGRAMA = NUMERO DO PROGRAMA
DO PROCESSO EM EXECUCAO, TIPO DA TEMPORIZ = CICLICA OU
COMUM

CLASSE = RAPIDA OU LENTA E PRIORIDADE ENTRE 0 E 4)

THEN

BEGIN

VARRE LISTA DO TIPO E CLASSE

IF EXISTE TEMPORIZACAO PARA O PROCESSO

THEN

BEGIN

LIBERA CELULA DE TEMPORIZACAO

REORDENA LISTAS

INDICA SUCESSO NA CHAMADA

```
                RETORNA
            END
        END
    ELSE
    BEGIN
        BUFALHA ^--- CODIGO E DADOS DA FALHA (ERRO NOS
                    PARAMETROS OU NAO EXISTE TEMPORIZ.)
        CALL FALHA
        RETORNA
    END
END                ; FIM DA PRIMITIVA CANTEMP

*** FIM DA FUNCAO CONTROLE DE TEMPORIZACOES
```

```
*** FUNCAO CONTROLE DE SINAIS
** AREA DE DEFINICAO
* DESCRICAO:
  * ESTA FUNCAO GERENCIA OS RECURSOS DE SINAIS
  * SUAS CELULAS E LISTAS
* CHAMADA POR:
  * CONTROLE DE INICIACAO, GERENCIA DE PROCESSOS
* CHAMA:
  * CONTROLE DE TEMPORIZACOES, GERENCIA DE PROCESSOS,
  * CONTROLE DE FALHAS E COMUNICACAO EXTERNA
* TAMANHO:
  * X (CODIGO) E Y (DADOS)
** AREA DE COMUNICACAO
.
.
.
** AREA DE DADOS
* CONSTANTES
.
.
.
* VARIAVEIS (DATA SEGMENT)
.
.
.
* SUB-ROTINAS (CODE SEGMENT)
.
.
.
* MACROS
.
.
.
```

** AREA DE COMANDOS

SINAIS:

BEGIN

IF INICIACAO

THEN

BEGIN

IF INICIACAO DO CONTROLE DE SINAIS

THEN

BEGIN

LIBERA CELULAS DE SINAIS CURTOS COLOCANDO-AS NA LISTA
DE CELULAS LIVRES

LIBERA CELULAS DE SINAIS LONGOS COLOCANDO-AS NA LISTA
DE CELULAS LIVRES

LIBERA CELULAS DA LISTA DE CORREIO

LIBERA CELULAS DA LISTA DE DIFUSAO INTERNA

LIBERA CELULAS DA LISTA DE TRANSMISSAO EXTERNA

COLOCA LISTAS COMO VAZIAS (EXCECAO DA LISTA DE CELULAS
LIVRES)

RETORNA

END

ELSE

BEGIN

; INICIACAO DA COMUNICACAO EXTERNA

COLOCA UMA CELULA DE SINAL LONGO COMO

BUFFER DE RECEPCAO EXTERNA

INDICA QUE NAO HA' SINAL SENDO TRANSMITIDO

CALL TESTE CIRC. DE COMUNICACAO EXTERNA

IF ERRO

THEN

INDICA FALHA NO CIRCUITO DE COMUNICACAO EXTERNA

ELSE

BEGIN

INDICA INICIACAO DA COMUNICACAO EXTERNA OK

CALL PROGRAMA CIRCUITO DE COMUNICACAO EXTERNA

RETORNA

END

END

ELSE

BEGIN

; PROCESSOS CHAMA

LIBERA CELULAS DE SINAIS LIGADAS AO PROCESSO INDICADO

RETORNA

END

* PRIMITIVAS DA FUNCAO CONTROLE DE SINAIS

SENDSIN:

BEGIN

; FUNCAO - ENVIA SINAL

; ENTRADA - PROGRAMA/PROCESSO/INSTANCIA DESTINOS, IDENTIDADE
 ; DO SINAL, PRIORIDADE DO SINAL, TIPO DO SINAL, TIPO
 ; DE TRANSMISSAO (EXT./INT., DESTINO DETERMINADO/
 ; DESTINO INDETER., COM ESPERA/SEM ESPERA, DIFUSAO
 ; EXT./CORREIO/DIFUSAO INT./SEM DESTINO ESPECIFICO),
 ; POSICAO DE EXECUCAO (NO CASO DE ESPERA), POSICAO DE
 ; EXECUCAO (TIME-OUT)

; SAIDA - OK

; ERRO (NAO HA' CELULA DE SINAL LIVRE, PARAMETROS
 ; ERRADOS, CIRCUITO DE TRANSM. EXTERNA COM DEFEITO)

IF PARAMETROS DE ENTRADA OK (PRIORID. DO SINAL, TIPO DO
 SINAL E TIPO DE TRANSMISSAO)

THEN

BEGIN

IF EXISTE CELULA DO TIPO PEDIDO LIVRE

THEN

BEGIN

ALOCA CELULA DO SINAL

COPIA DADOS PARA CELULA DO SINAL

IF SINAL EXTERNO

THEN

BEGIN

CALL ESTADO DA COMUNICACAO EXTERNA

IF ERRO

THEN

BEGIN

BUFALHA ^--- CODIGO E DADOS DO ERRO

CALL FALHA

RETORNA

END

ELSE

BEGIN

IF DESTINO INDETERMINADO

THEN

CALL PRODES

COLOCA SINAL NA LISTA DE SINAIS
 RELATIVA A SEU TIPO DE TRANSMISSAO

CALL PROCESSO (VERIFICA SE PROCESSO DESTINO
 ESTA' A ESPERA DE SINAL)

IF ESPERA

THEN

CALL PROCESSO (PEDE COLOCACAO DO PROCESSO
 EM EXECUCAO NA ESPERA DO SINAL SER
 RECEBIDO)

ELSE

BEGIN

INDICA SUCESSO NA CHAMADA

RETORNA

END

END

END

END


```

END
ELSE          ; ERRO
BEGIN
    BUFALHA ^--- CODIGO DE DADOS DO ERRO (FALTA DE CELULA
                LIVRE, PARAMETROS ERRADOS)
    CALL FALHA
    RETORNA
END
END          ; FIM DA PRIMITIVA SENDSIN

RECESIN:
BEGIN
; FUNCAO - RECEBE SINAL
; ENTRADA - DIFUSAO INTERNA/EXTERNA/CORREIO/COMUM, COM ESPERA/SEM
;           ESPERA, SINAL QUALQUER/GRUPO DE SINAIS (ATE' 6)
; SAIDA - OK (SINAL E' COPIADO NO BUFFER DE COMUNICACAO DO
;           PROCESSO E LIBERADO) OU NAO TEM SINAL
;           - ERRO (PARAMETROS ERRADOS)
IF PARAMETROS OK
THEN
BEGIN
    VERIFICA SE NA LISTA RELATIVA AO SEU PEDIDO EXISTE O
    SINAL PEDIDO OU SINAIS PEDIDOS
    IF EXISTE
    THEN
    BEGIN
        COPIA SINAL PARA BUFFER DE COMUNICACAO DO
        PROCESSO
        INDICA SUCESSO NA CHAMADA
        LIBERA CELULA DO SINAL
        CALL PROCESSO (VERIFICA SE ALGUM PROCESSO ESTA' A
        ESPERA DE QUE ESSE SINAL SEJA RECEBIDO)
        RETORNA
    END
    ELSE
    BEGIN
        ; NAO EXISTE SINAL
        IF ESPERA
        THEN
            CALL PROCESSO (PEDE COLOCACAO DE PROCESSO A
            ESPERA DE SINAL)
        ELSE
        BEGIN
            INDICA QUE NAO TEM SINAL
            RETORNA
        END
    END
END
END
ELSE          ; ERRO PARAMETROS
BEGIN
    BUFALHA ^--- CODIGO E DADO DO ERRO (PARAMETROS
                ERRADOS - CANTEMP)
    CALL FALHA
    RETORNA
END
END          ; FIM DA PRIMITIVA RECESIN

```

```

LIBESIN:
BEGIN
; FUNCAO - LIBERAR SINAIS ENVIADOS POR UM PROCESSO
; ENTRADA - NUMERO DE SINAIS (UM SINAL, ALGUNS, TODOS),
;           IDENTIDADE DOS SINAIS
; SAIDA - OK (SINAIS LIBERADOS, NAO HAVIA SINAIS)
;         - ERRO (PARAMETROS ERRADOS)
IF PARAMETROS OK
THEN
BEGIN
    OBTEM NUMERO DE SINAIS NO BUFFER DE COMUNICACAO
    RELATIVO AO PROCESSO
    VARRE LISTAS DE SINAIS PARA LIBERAR SINAIS ENVIADOS
    PELO PROCESSO EM EXECUCAO
    IF ENCONTROU SINAL
    THEN
    BEGIN
        LIBERA CELULA DO SINAL
        INDICA QUE LIBEROU SINAL
    END
    IF FIM DAS LISTAS
    THEN
    BEGIN
        IF LIBEROU SINAL
        THEN
            INDICA LIBERACAO DE SINAIS
        ELSE
            INDICA QUE NAO HOUVE LIBERACAO DE SINAIS
        RETORNA
    END
    ELSE
        VOLTA A VARRER LISTAS DE SINAIS
    END
ELSE
BEGIN
    BUFALHA ^--- CODIGO E DADOS DO ERRO (PARAMETROS
                ERRADOS NA CHAMADA DE LIBESIN)
    CALL FALHA
    RETORNA
END
END ; FIM DA PRIMITIVA LIBESIN

```

```

SAVESIN:
BEGIN
; FUNCAO - SALVAR SINAIS LIGADOS AO PROCESSO EM EXECUCAO
; ENTRADA - NUMERO DE SINAIS (UM/ALGUNS/TODOS), IDENTIDADE DOS
;           SINAIS
; SAIDA - OK (PRIMITIVA EXECUTADA)
;         - ERRO (NAO HAVIA SINAIS PEDIDOS, SINAL JA' MARCADO
;           PARA SAVE
;           ANTERIORMENTE E LIBERADO AGORA, PARAMETROS ERRADOS)
OBTEM NUMERO DE SINAIS E IDENTIDADE DE CADA SINAL NO BUFFER DE
COMUNICACAO RELATIVO AO PROCESSO
IF PARAMETROS DE ENTRADA OK
THEN

```

```
BEGIN
  DO WHILE NUMERO DE SINAIS      0
    VARRE LISTA DE SINAIS LIGADOS AO PROCESSO EM EXECUCAO
    PROCURANDO SINAL INDICADO
    IF ENCONTROU
      THEN
        BEGIN
          IF SINAL NAO MARCADO PARA SAVE
            THEN
              MARCA SINAL PARA SAVE E INDICA QUE SINAL FOI
              MARCADO
            ELSE
              DESMARCA SAVE ANTERIOR E INDICA LIBERACAO
              DO SINAL
          END
        ELSE
          INDICA QUE SINAL NAO EXISTE
          PROXIMO SINAL
        END
      END
    RETORNA
  END
ELSE
  BEGIN
    BUFALHA ^--- CODIGO E DADOS DA FALHA
    CALL FALHA
    RETORNA
  END
END
; FIM DA PRIMITIVA SAVESIN

*** FIM DA FUNCAO CONTROLE DE SINAIS
```

*** FUNCAO CONTROLE DE FALHAS
** AREA DE DEFINICAO
* DESCRICAO:
* ESTA FUNCAO PERMITE A GERENCIA DAS FALHAS PROVOCADAS
* PELOS PROCESSOS OU NUCLEO
* CHAMADA POR:
* CONTROLE DE INICIACAO, GERENCIA DE PROCESSOS, CONTROLE DE
* SINAIS, CONTROLE DE TEMPORIZACOES, SIST. OPERAC. E

USUARIO

* CHAMA:
* CONTROLE DE INICIACAO, DEPURADOR, SIST. OPERAC.
* TAMANHO:
* X (CODIGO) E Y (DADOS)
** AREA DE COMUNICACA
.
.
.
** AREA DE DADOS
* CONSTANTES
.
.
.
* VARIAVEIS (DATA SEGMENT)
.
.
.
* SUB-ROTINAS (CODE SEGMENT)
.
.
.
* MACROS
.
.
.

```

** AREA DE COMANDOS
FALHA:
BEGIN
  IF INICIACAO
  THEN
  BEGIN
    BUFALHA ^--- AUSENCIA DE FALHA
    CALL INFMOD (OBTEM MODO DE TRATAMENTO DE FALHAS)
    ATUALIZA MODTRAT
    APAGA ALARME DO PROCESSADOR
    CALL INFDEPU (VERIFICA SE EXISTE DEPURADOR)
    ATUALIZA DEPURADOR
    RETORNA
  END
  ELSE
  BEGIN
    OBTEM CODIGO DA FALHA EM BUFALHA
    IF FALHA = CLASSE 1
    THEN
    BEGIN
      IF MODTRAT = PARADA
      THEN
      BEGIN
        IF DEPURADOR EXISTE
        THEN
        BEGIN
          LIGA ALARME
          CALL PARAPRO
        END
        ELSE
          PISCA ALARME (FICA EM LOOP)
      END
      ELSE
      BEGIN
        LIGA ALARME
        CALL INFALSO (INFORMA FALHA QUE PROVOCA
          REINICIACAO AO S.O. PARA
          ARMAZENAMENTO EM MEMORIA
          DE MASSA)
        CALL REINIPR (REINICIA PROCESSADOR)
      END
    END
    IF FALHA = CLASSE 2
    THEN
    BEGIN
      IF MODTRAT = PARADA
      THEN
      BEGIN
        IF DEPURADOR EXISTE
        THEN
        BEGIN
          LIGA ALARME
          CALL PARAPRO
        END
        ELSE
          PISCA ALARME (LOOP)
      END
    END
  END

```

```

END
ELSE
BEGIN
    LIGA ALARME
    CALL INFALSO
    RESTAURA DADO DO NUCLEO PARA CONTINUAR OPERACAO
    NORMAL
    RETORNA
END
END
IF FALHA = CLASSE 3
THEN
BEGIN
    IF NIVTRAT = USUARIO
    THEN
        LIGA ALARME
        CALL PROCESSO (PEDE ESCALACAO DO PROCESSO
                        TRATADOR DE FALHAS DO PROGRAMA)
        ; NAO EXISTE PROCESSO TRATADOR DE FALHAS
        ; DO PROGRAMA
        CALL INFALSO (FORNECE DADOS DA FALHA ANTERIOR
                     PARA O S.O.)
        BUFALHA ^--- CODIGO E DADOS DA FALHA (FALTA DE
                    PROCESSO TRATADOR DE FALHAS)
        CALL INFALSO
        RESTAURA DADOS DO NUCLEO
        RETORNA
    END
    ELSE
    BEGIN
        IF MODTRAT = PARADA
        THEN
            BEGIN
                IF DEPURADOR EXISTE
                THEN
                    BEGIN
                        LIGA ALARME
                        CALL PARAPRO
                    END
                ELSE
                    PISCA ALARME
                END
            END
        ELSE
            BEGIN
                LIGA ALARME
                CALL INFALSO
                RESTAURA DADOS DO NUCLEO
                RETORNA
            END
        END
    END
END
END
BUFALHA ^--- CODIGO E DADOS DO ERRO (CODIGO DE FALHA NAO
RECONHECIDO)
IF MODTRAT = PARADA
THEN
BEGIN

```

```
IF DEPURADOR EXISTE
THEN
BEGIN
    LIGA ALARME
    CALL PARAPRO
END
ELSE
    PISCA ALARME
END
ELSE
BEGIN
    LIGA ALARME
    CALL INFALSO
    RESTAURA DADOS DO NUCLEO
    RETORNA
END
```

* PRIMITIVAS DA FUNCAO CONTROLE DE FALHAS

ALTERNIV:

BEGIN

; FUNCAO - ALTERA NIVEL DE TRATAMENTO DAS FALHAS CLASSE 3

; ENTRADA - NIVEL DE TRATAMENTO

; SAIDA - OK

;

ERRO (NAO HA' PROCESSO TRATADOR DE FALHAS DO PROGRAMA
VARRE LISTA DE PROCESSOS TRATADORES DE FALHA ATIVOS
IF EXISTE PROCESSO TRATADOR DE FALHA PARA O PROGRAMA DO
PROCESSO EM EXECUCAO

THEN

BEGIN

ALTERA NIVEL DE TRATAMENTO NA CELULA DO PROCESSO
EM EXECUCAO CONFORME PEDIDO
RETORNA

END

ELSE

BEGIN

INDICA NAO EXISTENCIA DE PROCESSO TRATADOR DE FALHAS
NO BUFFER DE COMUNICACAO DO PROCESSO
RETORNA

END

END ; FIM DA PRIMITIVA ALTERNIV

PARAPRO:

BEGIN

; FUNCAO - TRANSFERE CONTROLE DO PROCESSADOR PARA O DEPURADOR

; ENTRADA - NAO HA'

; SAIDA - NAO HA'

IF PRIPROEX = 0 OU 5

THEN

BEGIN

IF DEPURADOR EXISTE

THEN

BEGIN

LIGA ALARME

TRANSFERE CONTROLE PARA DEPURADOR

END

ELSE

PISCA ALARME

END

ELSE

BEGIN

BUFALHA ^--- CODIGO E DADOS DO ERRO (CHAMADA POR
PROCESSO NAO AUTORIZADO)

CALL FALHA

END

END ; FIM DA PRIMITIVA PARAPRO

INFALHA:

BEGIN

; FUNCAO - FORNECE OS DADOS DE FALHA AO PROCESSO TRATADOR DE
FALHAS

; ENTRADA - NAO HA'

; SAIDA - OK (DADOS DA FALHA)


```
        (NAO HA' FALHA)
IF PRIPROEX = 0
THEN
BEGIN
    IF EXISTE FALHA
    THEN
    BEGIN
        COPIA DADOS DA FALHA DE BUFALHA PARA O BUFFER DE
        COMUNICACAO DO PROCESSO TIPO 0
        RETORNA
    ELSE
    BEGIN
        INDICA QUE NAO HA' FALHA
        RETORNA
    END
END
ELSE
BEGIN
    BUFALHA ^--- CODIGO E DADOS DA FALHA (CHAMADA POR
    PROCESSO NAO AUTORIZADO)
    CALL FALHA
END
END ; FIM DA PRIMITIVA INFALHA
*** FIM DA FUNCAO CONTROLE DE FALHAS
**** FIM DO BLOCO DAS FUNCOES
```

**** BLOCO DAS INTERRUPTOES

```

** INTERRUPTAO DE RELOGIO DE TEMPO REAL (4 ms)
INTERRUPTAO 4 ms:
BEGIN
  INIBE INTERRUPTOES
  SALVA CONTEXTO DE QUEM ESTAVA EXECUTANDO
  IF PRIPROEX = 3 OU 4
  THEN
    BEGIN
      IF EXISTE PROCESSO TIPO 1 OU 2
      THEN
        BEGIN
          IF NUMERO DO PROGRAMA DO PROCESSO 1 OU 2
            NUMERO DO PROGRAMA DO PROCESSO 3 OU 4
          THEN
            BEGIN
              SUSPENDE PROCESSO QUE ESTAVA EM EXECUCAO
              SALVA CONTEXTO NA PILHA DE SALVAMENTO DE CONTEXTO
                (INCLUINDO WDT ATUAL)
              COLOCA PROCESSO 1 OU 2 COMO EM EXECUCAO
            END
          END
        END
      END
    END
  IF EXISTE TEMPORIZACAO RAPIDA
  THEN
    BEGIN
      DO WHILE EXISTE CELULA
        OBTEM CELULA
        DECREMENTA CONTADOR
        IF CONTADOR = 0
        THEN
          BEGIN
            OBTEM CELULA CORRESPONDENTE DO PROCESSO NA LISTA DE
            ESPERA POR TEMPORIZACAO
            COLOCA PROCESSO NO INICIO DA LISTA DE ATIVOS DE SUA
            PRIORIDADE
          END
        END
      PROXIMA CELULA DE TEMPORIZACAO
    END
  END
  IF EXISTE SINAL NA FILA EXTERNA PARA TRANSMISSAO
  THEN
    BEGIN
      IF MODULO DE COMUNICACAO EXTERNA LIVRE
      THEN
        BEGIN
          OBTEM PRIMEIRA CELULA DA LISTA
          COLOCA COMO SINAL EM TRANSMISSAO
          PROGRAMA MODULO DE COMUNICACAO EXTERNA
          PEDE TRANSMISSAO
        END
      END
    END
  END
  IF LISTA DE DIFUSAO INTERNA TEM SINAL
  THEN

```

```
BEGIN
  DO UNTIL FIM DA LISTA
    OBTEM CELULA DE SINAL
    DECREMENTA CONTADOR
    IF CONTADOR = 0
      THEN
        LIBERA CELULA
        PROXIMA CELULA
      END
    END
  END
  RETORNA CONTEXTO
  HABILITA INTERRUPCOES
  RETORNA
END ; FIM DA ROTINA DE INTERRUPCAO 4 ms
```

```

** INTERRUPTAO DE RELOGIO DE 1 s
INTERRUPTAO 1 s:
BEGIN
  INIBE INTERRUPTOES
  PARA WDT
  SALVA CONTEXTO DE QUEM ESTAVA EXECUTANDO
  IF EXISTE TEMPORIZACAO LENTA
  THEN
  BEGIN
    DO WHILE EXISTE CELULA
      OBTEM CELULA
      DECREMENTA CONTADOR
      IF CONTADOR = 0
      THEN
      BEGIN
        OBTEM CELULA DO PROCESSO CORRESPONDENTE A TEMPORIZ.
        COLOCA PROCESSO NO FIM DA LISTA DE ATIVOS DE SUA
        PRIORIDADE
      END
      PROXIMA CELULA
    END
  END
  IF LISTA DE DIFUSAO INTERNA TEM SINAL
  THEN
  BEGIN
    DO UNTIL FIM DA LISTA
      OBTEM CELULA DE SINAL
      DECREMENTA CONTADOR
      IF CONTADOR = 0
      THEN
        LIBERA CELULA
      PROXIMA CELULA
    END
  END
  END
  RETORNA CONTEXTO
  HABILITA INTERRUPTOES
  REATIVA WDT
  RETORNA
END ; FIM DA ROTINA DE INTERRUPTOES 1 s

```

```
** INTERRUPTAO DE RECEPCAO DE SINAL EXTERNO
INTERRUPTAO RECEXT:
BEGIN
  INIBE INTERRUPTOES
  PARA WDT
  SALVA CONTEXTO DE QUEM EXECUTAVA
  IF RECEPCAO OK
  THEN
  BEGIN
    OBTEM CELULA LIVRE
    COPIA SINAL DO BUFFER DE RECEPCAO PARA CELULA LIVRE
    COLOCA CELULA DO SINAL NA LISTA RELATIVA AO SEU TIPO
    PROGRAMA CIRCUITO DE RECEPCAO PARA NOVA RECEPCAO
    RETORNA CONTEXTO
    HABILITA INTERRUPTOES
    REATIVA WDT
    RETORNA
  END
  ELSE
  BEGIN
    BUFALHA ^--- CODIGO E DADOS DA FALHA
    CALL FALHA
    RETORNA CONTEXTO
    HABILITA INTERRUPTOES
    REATIVA WDT
    RETORNA
  END
END      ; FIM DA ROTINA DE INTERRUPT. DE RECEPCAO DO SINAL
          ; EXTERNO
```

```
** INTERRUPTAO POR ESTOURO DE WDT
INTERRUPTAO WDT:
BEGIN
  INIBE INTERRUPTOES
  SALVA CONTEXTO
  IF EXISTE CIRCUITO DE WDT
  THEN
  BEGIN
    BUFALHA ^--- CODIGO E DADOS DA FALHA
    SUSPENDE PROCESSO POR ERRO
    CALL FALHA
    PARA PROCESSO SUSPENSO POR ERRO E LIBERA SEUS RECURSOS
    PULA PARA VARRER (HABILITA INTERRUPTOES)
  END
  RETORNA CONTEXTO
  HABILITA INTERRUPTOES
  RETORNA
END ; FIM DA ROTINA DE INTERRUPTOES DE WDT
```

```
** INTERRUPTAO POR VIOLACAO DE MEMORIA
INTERRUPTAO VIOLMEM:
BEGIN
  INIBE INTERRUPTOES
  SALVA CONTEXTO
  IF EXISTE CIRC. PROT. MEM.
  THEN
  BEGIN
    BUFALHA ^--- CODIGO E DADOS DA FALHA
    SUSPENDE PROCESSO POR ERRO
    CALL FALHA
    PARA PROCESSO SUSPENSO POR ERRO
    LIBERA RECURSOS LIGADOS AO PROCESSO
    HABILITA INTERRUPTOES
    PULA PARA VARRER
  END
  RETORNA CONTEXTO
  HABILITA INTERRUPTOES
  RETORNA
END
; FIM DA ROTINA DE INTERRUPT. POR VIOLACAO
; DE MEMORIA
```

** INTERRUPTAO DE ESTOURO DE PILHA

INTERRUPTAO ESTPILHA:

BEGIN

INIBE INTERRUPTOES

SALVA CONTEXTO

IF EXISTE CIR. CONT. PILHA

THEN

BEGIN

BUFALHA ^--- CODIGO E DADOS DA FALHA

SUSPENDE PROCESSO POR ERRO

CALL FALHA

PARA PROCESSO SUSPENSO POR ERRO

LIBERA RECURSOS LIGADOS AO PROCESSO

HABILITA INTERRUPTOES

PULA PARA VARRER

END

RETORNA CONTEXTO

HABILITA INTERRUPTOES

RETORNA

END

; FIM DA ROTINA DE INTERRUPT. POR ESTOURO DE

; PILHA

**** FIM DO BLOCO DAS INTERRUPTOES

***** FIM DO NUCLEO

I.2. DESCRIÇÃO DE UM PROGRAMA DO USUÁRIO

```

***** PROGRAMA DE ALARMES DA CENTRAL
**** BLOCO COMUM
*** AREA DE DEFINICAO
** DESCRICAO:
* ESTE PROGRAMA RECEBE AS INFORMACOES DE ALARMES DOS DEMAIS
* PROGRAMAS DO PROCESSADOR LOCAL, FORNECE ESSAS INFORMACOES
AO
* CENTRALIZADOR DE LARMES E ATUALIZA DISPLAY DE ALARMES DO
* OPERADOR
** PROCESSOS:
* PROCESSO A - INICIACAO
* PROCESSO B - ATUALIZA DISPLAY DE ALARMES DO OPERADOR
* PROCESSO C - RECEBE ALARMES E INFORMA CENTRALIZADOR
* PROCESSO D - TRATADOR DE FALHAS DO PROGRAMA
** CARACTERISTICAS:
* TAMANHO - X (CODIGO) E Y (DADOS)
* LINGUAGEM - PL/M
* PROCESSADOR - INTEL 8085
** VERSAO:
* AUTORES - CARLOS CESAR FERREIRA ARAUJO
* DATA - 10/12/1983
* LOCAL - CPqD (TELEBRAS)
* PROJETO - X

*** AREA DE COMUNICACAO
** BLOCO DE COMUNICACAO GLOBAL
INCLUDE BLOCOGB

*** AREA DE DADOS
** CONSTANTES (CODE SEGMENT)
.
.
.
** VARIAVEIS (DATA SEGMENT)
.
.
.
** SUB-ROTINAS (CODE SEGMENT)
.
.
.
** MACROS
.
.
.
**** FIM DO BLOCO COMUM

```

```
**** BLOCO DOS PROCESSOS
*** PROCESSO A
** AREA DE DEFINICAO
* DESCRICAO:
  * ESTE PROCESSO INICIA O PROGRAMA
* TIPO DO PROCESSO
  * TIPO 1 (INICIACAO)
** AREA DE COMUNICACAO
.
.
.
** AREA DE DADOS
* CONSTANTES
.
.
.
* VARIAVEIS (DATA SEGMENT)
.
.
.
* SUB-ROTINAS (CODE SEGMENT)
.
.
.
* MACROS
.
.
.
```

```
** AREA DE COMANDOS
PROCESSO A:
BEGIN
  PROCINIC:
  BEGIN
    BUFCOM1 ^--- NUMERO DO PROCESSO D, NUMERO DA INSTANCIA (UNICA),
                TIPO 0, WDT = 20 ms, POSICAO DE EXECUCAO = PROCD.
    CALL STARTPR
    BUFCOM1 ^--- NUMERO DO PROCESSO B, NUMERO DA INSTANCIA (UNICA),
                TIPO 2, WDT = 40 MS, POSICAO DE EXECUCAO = PROCB.
    CALL STARTPR
    BUFCOM1 ^--- NUMERO DO PROCESSO C, NUMERO DA INSTANCIA (UNICA),
                TIPO 3, WDT = 100 MS, POSICAO DE EXECUCAO = PROCC.
    CALL STARTPR
    CALL INFPROG (INFORMA AO S.O. PROGRAMA INICIADO)
    CALL STOPROC
  END
END
*** FIM DO PROCESSO A
```

```
*** PROCESSO B
** AREA DE DEFINICAO
* DESCRICAO:
  * ESTE PROCESSO RECEBE ALARMES DOS DEMAIS PROGRAMAS DO
USUARIO
  * DO PROCESSADOR LOCAL E OS INFORMA AO CENTRALIZADOR
* TIPO DO PROCESSO:
  * 2
** AREA DE COMUNICACAO
  .
  .
  .
** AREA DE DADOS
* CONSTANTES
  .
  .
  .
* VARIAVEIS (DATA SEGMENT)
  .
  .
  .
* SUB-ROTINAS (CODE SEGMENT)
  .
  .
  .
* MACROS
  .
  .
  .
```

```
** AREA DE COMANDOS
PROCESSO B:
BEGIN
  PROCB:
  BEGIN
    BUFCOM2 ^--- NIVEL DE TRATAMENTO = USUARIO
    CALL ALTENIV
    BUFCOM2 ^--- RECEPCAO COMUM, COM ESPERA, 2 SINAIS,
                ALARME - POSICAO DE EXECUCAO = PROCB1
                ENVALP - POSICAO DE EXECUCAO = PROCB2
    CALL RECESIN
  END

  PROCB1:
  BEGIN
    OBTEM DADOS DO SINAL
    OBTEM ORIGEM
    ATUALIZA ALARMES DO PROGRAMA ORIGEM COM DADOS DO SINAL
  END

  PROCB2:
  BEGIN
    OBTEM ALARMES DO PROCESSADOR LOCAL
    BUFCOM2 ^--- PREENCHE COM DADOS DE DESTINO E ALARMES, ENVIO
                SEM ESPERA, EXTERNO, DESTINO JA' DETERMINADO,
                SINAL COMUM, TIPO 1
    CALL SENDSIN
  END
END
END
*** FIM DO PROCESSO B
```

```
*** PROCESSO C
** AREA DE DEFINICAO
* DESCRICAO:
  * ESTE PROCESSO ATUALIZA O PAINEL DE DISPLAYS DO OPERADOR
* TIPO DO PROCESSO:
  * 3
** AREA DE COMUNICACAO
  .
  .
** AREA DE DADOS
* CONSTANTES
  .
  .
* VARIAVEIS (DATA SEGMENT)
  .
  .
* SUB-ROTINAS (CODE SEGMENT)
  .
  .
* MACROS
  .
  .
  .
```

```
** AREA DE COMANDOS
PROCESSO C:
BEGIN
  PROCC:
  BEGIN
    BUFCOM3 ^--- NIVEL DE TRATAMENTO = USUARIO
    CALL ALTENIV
    APAGA DISPLAYS DE ALARMES DO OPERADOR
    BUFCOM3 ^--- NUMERO DO PROCESSO C, NUMERO DA INSTANCIA (UNICA),
              CLASSE LENTA, TIPO CICLICA, POSICAO DE EXECUCAO =
              PROCC1.

    CALL PEDTEMP
  END

  PROCC1:
    OBTEM RESUMO DE ALARMES DO PROCESSADOR LOCAL
    ATUALIZA DISPLAY DO OPERADOR
  END
END
*** FIM DO PROCESSO C
```

```
*** PROCESSO D
** AREA DE DEFINICAO
* DESCRICAO
  * PROCESSO TRATADOR DE FALHAS
* TIPO DO PROCESSO:
  * 0
** AREA DE COMUNICACAO
  .
  .
  .
** AREA DE DADOS
* CONSTANTES
  .
  .
  .
* VARIAVEIS (DATA SEGMENT)
  .
  .
  .
* SUB-ROTINAS (CODE SEGMENT)
  .
  .
  .
* MACROS
  .
  .
  .
```



```
** AREA DE COMANDOS
PROCESSO D:
BEGIN
  PROCD:
  BEGIN
    CALL INFALHA
    OBTEM CODIGO DA FALHA
    IF ESTOURO DE WDT
    THEN
      BEGIN
        BUFCOMO ^--- NUMERO DO PROCESSO PARADO POR ERRO, NUMERO DA
                    INSTANCIA, TIPO DO PROCESSO, NOVO WDT = ANTIGO
                    + 10 ms, POSICAO DE EXECUCAO INICIAL

        CALL STARTPR
      END
    ELSE
      CALL PARAPRO
    END
  END
END
*** FIM DO PROCESSO D

**** FIM DO BLOCO DE PROCESSOS

***** FIM DO PROGRAMA
```

I.3. BLOCO DE COMUNICAÇÃO GLOBAL

```

BLOCOGB:
BEGIN
** NOME DOS PROGRAMAS
PROGRAMA BRF ; ALARMES
PROCESSOS A, B, C, D (TRATADOR DE FALHAS)
INSTANCIA UNICA
.
.
** NOME DOS SINAIS
ALARBI
ENVALP
.
.
** PRIMITIVAS DO NUCLEO
STARTPR, STOPROC, ... EXTERNAL
** PRIMITIVAS DO S.O.
INFDEP, INFMOD, ... EXTERNAL
** BLOCOS DE COMUNICACAO
BUFCOMO, ... EXTERNAL
** ENDERECOS HARDWARE
ALARMEPROC, INIRAM, FIMRAM, WDT, ...
** SUB-ROTINAS
.
.
** MACROS
.
.
END ; FIM DO BLOCO DE COMUNICACAO GLOBAL

```

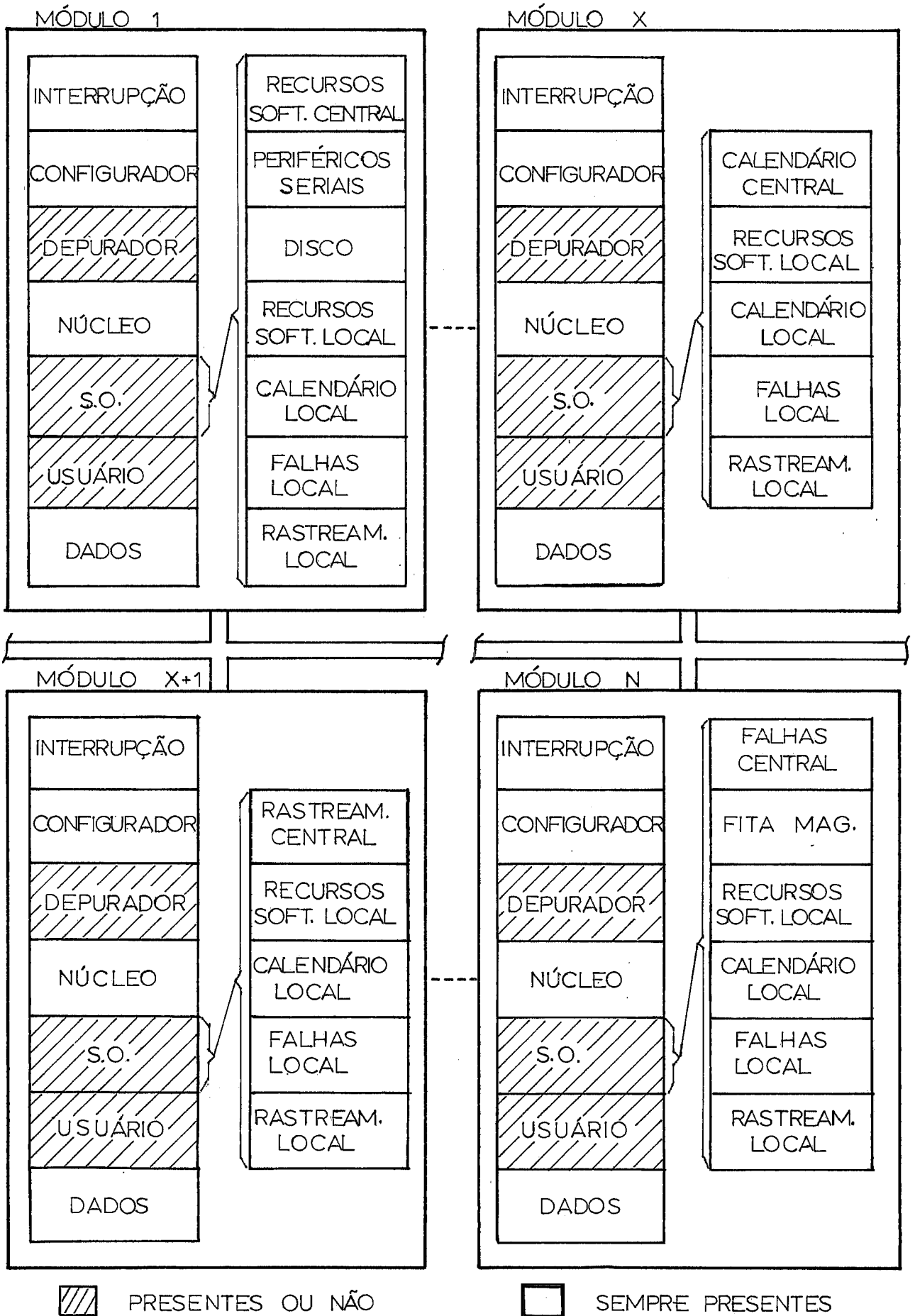
ANEXO IIDESCRIÇÃO DO SISTEMA OPERACIONAL

O Sistema Operacional a ser descrito é um conjunto de programas do qual fazem parte o depurador e o configurador. Além destes dois programas, o Sistema Operacional é constituído de programas para gerência de recursos "Software", gerência de tempo (calendário), gerência de rastreamento, gerência de falhas, testes e alarmes, gerência de periféricos, tanto seriais (terminais de vídeo, impressoras) como de memória de massa (fitas magnéticas, discos, disquetes).

Alguns desses programas, ou parte dos mesmos, se encontram em todos os processadores, enquanto outros se encontram em apenas um ou alguns processadores. Certos programas possuem além de um controle local (em cada processador), um controle central (em certos casos esse controle central pode ser duplicado para efeitos de confiabilidade).

O acesso a esses programas pode ser feito via primitivas para programas do mesmo processador ou via sinais para programas de outros processadores. Estes fatos permitem que a comunicação dentro de um mesmo processador seja feita de forma rápida e segura (via primitivas) e que os recursos do Sistema Operacional possam ser compartilhados por programas em diferentes processadores sem a necessidade da duplicação de recursos. Algumas primitivas do Sistema Operacional são de uso exclusivo do Núcleo.

A Figura (II.1) apresenta uma visão esquematizada do projeto, fornecendo a situação do Sistema Operacional dentro do mesmo.



FIGURA(1.1)

II.1. CONFIGURADOR

O Configurador esta' presente em todos os módulos do sistema. Ele estabelece a configuração inicial do processador local no qual esta' presente. Fornece ao Núcleo o número de programas existentes no processador local, os ponteiros dos processos de iniciação de cada programa, os limites das áreas de dados e código de cada programa (no caso de existir proteção de memória) e a configuração do "Hardware" local, ou seja, indica a existência ou não do circuito de proteção de memória, circuito de controle das pilhas, circuito de controle do WDT e circuito de comunicação externa. Informa também ao Núcleo a existência ou não do depurador e o modo de tratamento de falhas no processador local (com ou sem parada no depurador).

O Configurador ocupa as primeiras posições de código após as rotinas de tratamento das interrupções do sistema. O "reset" ocasiona uma chamada ao configurador. Este verifica a existência do depurador. Caso o depurador exista o configurador transfere o controle do processador local ao depurador. Caso não exista transfere o controle para o núcleo (função iniciação).

Função:

Fornecer a configuração do módulo local

Chamada por:

Núcleo
"reset"

Chama:

Depurador
Núcleo

Primitivas

As primitivas são de uso restrito do núcleo. A passagem dos dados e' feita através da área de comunicação BUFNUC.

INFHARD - fornece ao núcleo a configuração do hardware do módulo local.

Entrada - não ha'

Saída - indica existência ou não dos circuitos de proteção de memória, controle de pilhas, controle de WDT e comunicação externa.

INICONF - fornece o número de programas, ponteiros dos processos de iniciação de cada programa e limites das áreas de dados e código de cada programa.

Entrada - não ha'

Saída - número de programas, ponteiros e limites.

INFMOD - indica o modo de tratamento das falhas detectadas no módulo local. Se ao detectar uma falha o núcleo deve ou não transferir o controle ao depurador.

Entrada - não ha'

Saída - indicação do modo (com ou sem parada).

INFDEP - indica existência ou não do depurador no módulo local.

Entrada - não ha'

Saída - indicação de existência ou não do depurador.

A interface estabelecida entre o configurador e o núcleo, assim como a forma de implementação dos mesmos permitem a existência, caso a aplicação necessite, de carga dinâmica de programas (junto com a carga de programas deve ser provido um gerenciamento de memória).

II.2. DEPURADOR

O depurador é um programa que possibilita a verificação da condição de funcionamento de outros programas do módulo local. O depurador pode ou não existir em um determinado módulo, e sua condição de existência deve ser declarada no configurador. Através do depurador condições de erro podem ser verificadas e repetidas pelo usuário.

Quando o depurador existe em um módulo, todas as falhas quando detectadas pelo núcleo podem provocar, caso o usuário deseje, a transferência do controle do processador desse módulo para o depurador (as condições que provocaram a falha ficam mantidas).

Existe uma interface serial em cada módulo. A essa interface serial pode ser ligado um terminal de vídeo, através do qual o

usuário se comunica com o depurador. Existe uma interrupção em cada módulo local destinada ao depurador, possibilitando a implementação de seus comandos.

Função:

Depurar programas do módulo local

Chamada por:

Configurador

Núcleo

Interrupção do depurador

Chama:

Pode provocar a execução de qualquer procedimento (GO X)

Comandos:

- a) G x,y,n - este comando permite a execução via comando do depurador de um determinado procedimento.

G - identificador do comando

x - endereço inicial de execução

y - endereço final de execução (se omitido provoca a execução de somente uma instrução)

n - número de execuções

- b) DM x,y,c - mostra posições de memória de x a y no formato indicado em c. O formato pode ser hexa ou mnemônico. Se y for omitido mostra uma posição indicada por x. Se c for omitido mostra as posições em hexa.

- c) MM x,c,s - modifica as posições de memória a partir de x com os dados fornecidos na sequência s. O formato dos dados e' indicado por c. Pode ser hexa ou mnemônico (o default e' hexa).

- d) DR x - mostra registro x. Se x não for indicado mostra todos os registros.

- e) MR x,s - modifica registro x com valor fornecido em s. Se for fornecido mais de um valor em s modifica os registros seguintes conforme uma ordem estabelecida.

- f) DT c - permite que um conjunto de dados armazenados no buffer de trace (ate' 256 bytes) seja mostrado no formato indicado em c (hexa ou mnemônico).

- g) I l,x,y,c - permite a impressão de posições de memória, registros da área de trace no formato indicado em c (hexa ou mnemônico).

- h) S x,y - soma números hexa x e y

i) D x,y - subtrai números hexa x e y.

Todos os comandos propostos são simples, assim como também o e' a forma de implementá-los. Algo mais sofisticado como um depurador central (evita deslocamentos de terminais) e comandos de execução controlada podem ser implementados nessa estrutura.

II.3. GERÊNCIA DE RECURSOS "SOFTWARE"

Este programa controla os módulos existentes e os vários programas existentes em cada módulo. E' formado por uma parte central presente em um único módulo e uma parte local presente em todos os módulos.

O gerenciador central obtem do disco os dados que indicam os módulos existentes e as suas condições (em serviço, bloqueado pelo operador, bloqueado por falha ou não existente), mantém controle sobre essas condições, fornece esses dados aos gerenciadores locais dos módulos existentes e faz interface com o operador para modificação de condição de módulo.

O gerenciador local após receber os dados do gerenciador central, obtem do configurador os programas residentes em seu módulo e se comunica com os demais módulos existentes, para fornecer os seus dados e receber os dados dos outros. Isto permite a um módulo saber os programas existentes nos demais módulos do sistema. Periódicamente esses dados são verificados. Informa aos processos do módulo local ou ao núcleo sobre os módulos que contem determinado programa.

Função:

Gerência dos recursos "Software" do sistema

Chamada por:

usuário (via primitiva)

núcleo (via primitiva)

S.O. (via sinais - demais gerenciadores de recurso software)

Chama:

S.O. (via sinais - demais gerenciadores de recurso software)

configurador

Primitivas

PRODES - permite ao núcleo ou usuário obter os módulos que contem determinado programa (envio de sinais).

Entrada - número do programa

Saída - número de módulos e número de cada módulo que contem o programa

INFPROC - permite ao núcleo sinalizar para o gerenciador todas as vezes que um processo de programa e' ativado ou parado.

Entrada - número do programa, condição do processo

Saída - não ha'

O sistema como proposto permite a carga dinâmica de programas com pequenas alterações.

II.4. GERENCIADOR DE PERIFÉRICOS

Este programa e' responsável pelo controle dos diversos meios de entrada/saída existentes no sistema. E' constituído por um conjunto de programas, cada qual controlando um tipo de periférico e presente em um determinado módulo.

Um programa, ao necessitar realizar uma operação de entrada/saída em determinado periférico, pede envio de um sinal para o programa que controla o periférico. O módulo destino pode ser determinado pelo próprio programa ou pode ser deixado a cargo do núcleo sua determinação. Ao pedir o envio do sinal o programa deve fornecer o endereço lógico do periférico (nome do programa destino), o tipo de ação (nome do processo destino), o arquivo (nome da instância destino), os dados (caso existam) e os demais dados necessários para envio de sinais.

O programa central, após receber o sinal, realiza a operação pedida e caso haja resposta a envia ao programa solicitante diretamente. As interfaces externas do programa central de controle de um determinado periférico so' podem ser acessadas via sinais. O programa central e' responsável pelas interações diretas com os periféricos (inclui o driver do periférico-interface com o "hardware").

Os programas centrais de controle de periféricos podem ser subdivididos em dois grupos. O primeiro grupo e' constituído pelos programas centrais de controle dos periféricos de memória de massa. O segundo grupo e' constituído pelos programas centrais de controle de periféricos seriais.

Os programas de controle de memória de massa gerenciam as fitas magnéticas, discos e disquetes. Podem realizar, a pedido do usuário, operações de criação, deleção, leitura, escrita e atualização de arquivos. Além destas operações implementam também interfaces com o operador para manutenção. Os programas de controle de periféricos seriais gerenciam os terminais de vídeo e impressoras. Podem realizar operações de escrita e leitura, além das interfaces com o operador para manutenção.

Função:
Gerenciador de periféricos de E/S

Chamada por:
usuário
operador

Chama:
usuário
operador

Primitivas

-

II.5. GERENCIADOR DE TEMPO

Este programa é constituído de duas partes. A primeira é um gerenciador local, existente em todos os módulos, responsável pela manutenção do calendário no módulo local. Este calendário é constituído da hora/minuto, dia, mês, ano e dia da semana, além da indicação de feriado. O gerenciador local checa os seus dados periodicamente com os demais gerenciadores. A segunda parte é um gerenciador central responsável pela manutenção do calendário, além da manutenção de uma agenda e da interface com o operador. A agenda são procedimentos colocados na forma de sinal, que podem ser pedidos pelo operador ou pelo sistema e servem para ativar procedimentos a determinada hora. A interface com o operador serve para interrogação ou modificação da agenda ou calendário

Função:
Gerenciador de calendário, agenda

Chamada por:
usuário
operador

Chama:
usuário
operador

Primitivas

INFCAL - esta primitiva permite a um usuário obter o calendário (data, hora) do módulo local.

Entrada - não ha'

Saída - dados do calendário

II.6. GERENCIADOR DE FALHAS, ALARMES E TESTES

Este programa e' dividido em duas partes. A primeira parte e' um controlador local, existente em cada módulo, responsável pelo recolhimento de falhas e alarmes nesse módulo. A segunda parte e' um gerenciador central responsável pelo recolhimento (via sinal) das condições de falha e alarme em todos os módulos, pela manutenção de um arquivo "LOG" em memória de massa, pela atualização de um painel de alarmes central (visual e sonoro) e interface com o operador (interrogação de falhas).

Função:

Gerenciar falhas, alarmes e testes

Chamada por:

usuário (via primitivas ou sinais)
operador

Chama:

memória de massa
usuário (via primitiva)
operador

Primitivas

INFALSO - permite ao usuário ou núcleo informar suas falhas ao sistema operacional.

Entrada - dados da falha

Saída - não ha'

II.7. GERENCIADOR DE RASTREAMENTO

Este programa e' responsável pela visualização dos sinais trocados por processos que estejam marcados para rastreamento. E' formado por uma parte local presente em todos os módulos e uma

parte central única. A parte local recebe os pedidos de alteração da condição de rastreamento do central, pede alteração ao núcleo e recebe os sinais rastreados, enviando-os ao central. A parte central por sua vez implementa a interface com o operador via um terminal serial ligado ao módulo local.

Função:
Gerenciar sinais rastreados

Chamada por:
operador
usuário

Chama:
núcleo

Primitivas

-

BIBLIOGRAFIA

- 1 - Donovan, J.J. - "Systems Programming", McGraw-Hill LTD., New York, USA - 1972
- 2 - Guimarães, C.C. - "Princípios de Sistemas Operacionais", Editora Campus Ltda., Rio de Janeiro, Brasil - 1980
- 3 - Hansen, P.B. - "Operating Systems Principles", Prentice Hall Inc., New Jersey, USA - 1973
- 4 - Hansen, P.B. - "The Architecture of Concurrent Programs", Prentice Hall Inc., New Jersey, USA - 1977
- 5 - Madnick, S.E. - Donovan, J.J. - "Operating Systems", McGraw-Hill LTD., New York - 1974
- 6 - Segre, L. - Mendes, S. - Kimer, C. - "LPC: Uma Linguagem para Programação Concorrente", Relatório Técnico COPPE/ UFRJ - DCES/UFSCa, Rio de Janeiro, Brasil - 1982
- 7 - C.C.I.T.T. - "Introduction to CHILL", Manual
- 8 - Prorok, J.G. - Reis, L.A. - "Software Básico de uma Central Telefônica TROPICO-R", 3o. Simpósio de Software Básico para Micros, Rio de Janeiro, Brasil - Dezembro/1983
- 9 - Brinch Hansen, P. - "Edison: A Multiprocessor Language", Software Practice and Experience, Vol. 11, n. 4, Abril/1981 - páginas: 323-414
- 10 - Cook, P.R. - "*MOD: A Language for Distributed Programming", IEEE Transaction on Software Engineering, Vol. SE-6, n.6, Novembro/1980 - páginas: 563-571
- 11 - Silberschatz, A. - "Communication and Synchronization in Distributed Systems", IEEE Transactions on Software Engineering, Vol. SE-5, n.6, Novembro/1979 - páginas: 543 - 546
- 12 - Liskov, B. H. - Snyder, A. - "Exception Handling in CLU", IEEE Transactions on Software Engineering, Vol. SE-5, n.6, Novembro/1979 - páginas: 546 - 558
- 13 - Welsh, J. - Lister, A. - "A Comparative Study of Task Communication in ADA", Software Practice and Experience, Vol. 11, n. 3, Março/1981 - páginas: 257 - 290
- 14 - Hoare, C. A. R. - "Communication Sequential Processes", Communication of the ACM, Vol. 21, n.8, Agosto/1978 - páginas: 666 - 676
- 15 - Rekdal, K. - "CHILL: The Standard Language for Program-

- ming SPC Systems", IEEE Transactions on Communications, Vol. COM-30, n.6, Junho/1982 - páginas: 1318 - 1328
- 16 - Lawson, D.A.- "A New Software Architecture for Switching Systems", IEEE Transactions on Communications, Vol. COM-30, n. 6, Junho/1982 - páginas: 1281 - 1289
- 17 - Carreli, C.- Roche, D.J. - "C.C.I.T.T. Languages for SPC Switching Systems", IEEE Transactions on Communications, Vol. COM-30, n. 6, Junho/1982 - páginas: 1304 - 1309
- 18 - Penney, B.K.- Williams, J.W.J. - "The Software Architecture for a Large Telephone Switch", IEEE Transactions on Communications, Vol. COM-30, n. 6, Junho/1982 - páginas: 1269 - 1378
- 19 - Schutz, H.A. - "On the Design of a Language for Programming Real-Time Concurrent Processes", IEEE Transactions on Software Engineering, Vol. SE-5, n. 3, Maio/1979 - páginas: 248 - 255
- 20 - Brinch Hansen, P. - "Specification and Implementation of Mutual Exclusion", IEEE Transactions on Software Engineering, Vol. SE-4, n. 5, Setembro/1978 - páginas: 365 - 370
- 21 - Brinch Hansen, P. - "Distributed Processes: A Concurrent Programming Concept", Communications of the ACM, Vol. 21, n. 11, Novembro/1978 - páginas: 934 - 941
- 22 - Thorelli, L.E. - "A Microprogrammed System Kernel", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 135 - 142
- 23 - Muhlemann, K.- "Towards an Implementation of ADA Rendezvous Synchronization", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 289 - 302
- 24 - Young, S.J. - "Inter-process Communication Primitives for DSM Multiprocessors", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 327 - 332
- 25 - Istavrinos, P. - "Mapping of the ADA Rendezvous Concept onto a Multicomputer System", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 333 - 340
- 26 - Kacsuk, P. - "A Data Driven Emulator Module Based on a Concurrent Pascal Oriented Multiple Microprocess System", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 371 - 379
- 27 - Jul, E. - Schneider, H. - "Concurrent Pascal on the Intellect MDS", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 381 - 391

- 28 - Nielsen, P.M. - "A Multi-Processor for Multi-Program Experiments", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 393 - 397
- 29 - Conte, G. - "TOMP80: A Multiprocessor Prototype", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 401 - 410
- 30 - Krings, L. - "An Approach to Performance Measuring in Multiprocessor Systems with Time-Shared Buses", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 411 - 419
- 31 - Andreassen, J.K. - "A Multiprocessor Structure for Real-Time Processing", Euromicro Symposium, Paris, França, Setembro/1981 - páginas: 455 - 463
- 32 - Barbosa, V.C. - "Uma Proposta para Estender o Sistema Pascal USCD a Processamento Concorrente", Tese de Mestrado, COPPE/UFRJ, Agosto/1982
- 33 - US -DoD - "Reference Manual for the ADA Programming Language", 1983
- 34 - Hoare, C.A.R. - "Monitors: An Operating System Structuring Concept", Communications of the ACM, Vol. 17, n. 10, Outubro/1974 - páginas: 549 -557
- 35 - Brinch Hansen, P. - "The Programming Language Concurrent Pascal", IEEE Transactions on Software Engineering, Vol. SE-1, n. 2, Junho/1975 - páginas: 199 - 207
- 36 - US - DoD - "An Overview of ADA", Software Practice and Experience, Vol. 10, n. 2, Outubro/1980 páginas: 223-404
- 37 - Wirth- "Modula:A Language for Modular Multiprogramming", Software Practice and Experience, Vol. 7, n.3, Novembro/1977 - páginas: 212 - 310