UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIADA COMPUTAÇÃO

CARLOS EURICO PITTAS DO CANTO

Um sistema em tempo real multiprogramado para construção de sistemas de controle distribuído

Dissertação apresentada como requisito parcial para à obtenção do grau de Mestre em Ciências da Computação.

Prof. Dr. Cláudio Walter
Orientador

Prof. Dr. Simão Sirineo Toscani
Co-orientador

Porto Alegre, Janeiro de 1987.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

CANTO, Carlos Eurico Pittas do

Um sistema em tempo real multiprogramado para construção de sistemas de controle distribuído – Porto Alegre: CPGCC da UFRGS, 1987.

? f. il.

Dissertação (mestrado) Universidade Federal do Rio Grande do Sul. Curso de Pós-graduação em Ciências da computação. Porto Alegre, BR-RS, Janeiro de 1987. Orientador: Cláudio Walter.

- 2. 3.

I. Cláudio Walter. II. Título.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ILUSTRAÇÕES

RESUMO

Este trabalho descreve um modelo de arquitetura para sistemas de controle distribuído, o qual é usado para implementação de um sistema, utilizando recursos disponíveis em instalações convencionais. E detalhada a implementação e o seu núcleo, o qual é o componente encarregado de acrescentar ao ambiente original as características desejadas.

A multiprogrammed real-time system for building distributed control systems

This work describes a distributed control system architecture model used to define an implementation based on a nucleus. The implementation uses recourses normally found is conventional installations. The work also gives a detailed description of the system structure and the nucleus, which provides the required features.

1. INTRODUÇÃO

Este trabalho apresenta uma contribuição as pesquisas desenvolvidas pelo Grupo de Controle de Processos na área de Controle Distribuído. Apresentam-se os conceitos de sistemas de tempo real, sistemas de controle distribuído e propõe-se um modelo para tais sistemas sendo descrita uma implementação do mesmo.

O trabalho é organizado em nove capítulos abaixo resumidos.

No segundo capítulo, apresentamos os conceitos básicos e a organização de sistemas de tempo real.

No terceiro capítulo, expomos os sistemas de controle distribuído e as funções básicas de tais sistemas.

No quarto capítulo, consideramos o modelo para sistemas de controle distribuído. O modelo é estruturado em níveis e baseado nos conceitos de trocas de mensagens e objetos. Também é apresentado o modelo básico de referência proposta pela ISO para interconexões de sistemas abertos sendo analisado o possível relacionamento entre os dois modelos.

No quinto capítulo, descrevemos o projeto de implementação de um ambiente para construção de sistemas de controle distribuído. São apresentadas as características funcionais desejadas a estrutura do sistema e necessidades em termos de uma linguagem para aplicações de tempo real.

No sexto capítulo, apresentamos o núcleo do sistema e as diretivas mínimas necessárias ao suporte de aplicação desejada.

No sétimo capítulo, expressamos a implementação efetuada e suas características principais.

No oito capítulo, definimos as estruturas de dados do núcleo e descrevemos suas diretivas.

Finalmente, no capítulo 9, são apresentadas as conclusões e algumas considerações sobre desenvolvimentos futuros.

2. SISTEMAS DE TEMPO REAL

Um sistema pode ser definido como um conjunto de unidades funcionais distintas que interagem, integradas com um ambiente, para alcançar um objetivo comum / KLI 69 /.

Os sistemas de computação constituem-se numa parcela do universo de sistemas. Estes sistemas tem por finalidade fornecer ambientes simulados que possam representar sistemas físicos e abstratos existentes, de modo que seus usuários, utilizando ferramentas e mecanismos fornecidos, possam utilizar-se deles para resolução de problemas.

O termo tempo real abriga uma ampla gama de sistemas computadores (e outros), que compartilham das características comum de que os resultados de alguma operação são regidos por restrições impostas pelo mundo “real” externo ao sistema. A gama de restrições de tempo é bastante grande, desde microssegundos (como o processamento digital de sinais) até meses (num sistema de planejamento estratégico de uma empresa ou sistemas de coleta de dados sísmicos).

Neste trabalho nos deteremos na faixa intermediária da escala de tempos de reposta onde concentra-se a maioria das aplicações de controle de processos industriais. Evento que ocorrem mais do que um milhão de vezes por segundo são muito rápidos para os microprocessadores atuais sem um hardware auxiliar. Eventos no outro extremo da faixa não apresentam problemas, ou no caso de apresentarem, estes estão mais relacionados com a confiabilidade do armazenamento das informações, do que com a rapidez de processamento.

2.1 Os dois tipos de tempo real

O termo “tempo real” é usado para descrever dois tipos bem diferenciados de sistemas. Na sua utilização mais antiga, agora em desuso, era aplicado em sistemas tais como reserva de passagens áreas e centro de chaveamento de mensagens. Estes sistemas eram chamados de tempo real porque tinham um

operador sentado em frente a um terminal esperando por uma resposta. Atualmente estes sistemas são mais corretamente chamados de “interativos”. Um outro termo antigo para este tipo de sistema era “on-line” / SAV85 /.

Outro tipo de sistemas de tempo real, que se tornou usual após o surgimento dos minicomputadores e microcomputadores, corresponde aos sistemas de controle de processos. Nestes sistemas os computadores estão diretamente controlando e/ou monitorando algum processo externo.

2.1.1 Tipos de tempo real

Existem pelo menos três medidas de tempo as quais podem ser aplicadas aos sistemas de tempo-real: tempo de resposta, tempo de sobrevivência e desempenho ou banda passante.

- Tempo de resposta: é um tempo que o sistema usa para reconhecer e responder a um evento externo. Esta é uma medida mais importante em aplicações de controle pois se os eventos não forem manipulados dentro de um intervalo de tempo aceitável o processo controlado pode sair de controle.
- Tempo de sobrevivência: é o tempo durante o qual os dados estarão prontos esperando para serem usados. Não é o mesmo que tempo resposta: um dado pode estar válido por um breve tempo sem requerer uma resposta rápida, ou então pode permanecer válido por um longo período após alguma resposta o ter usado.
- Desempenho (ou throughput): é o número total de eventos que o sistema pode manipular em um dado intervalo de tempo. Por exemplo, um controlador de comunicação pode ter seu desempenho expresso em caracteres por segundo.
- Banda passante: quando o desempenho é medido em bits por segundo, como em aplicações de processamento de sinais, ele pode ser relacionado com a noção de banda passante.

2.2 Aplicações típicas

Mesmo excluindo as antigas aplicações de sistemas on-line ou tempo compartilhado os sistemas de tempo real abrangem uma larga gama de aplicações. Muitas destas aplicações são os sistemas dedicados (ou embedded system) nos quais o computador faz parte de um sistema maior e muitas vezes nem é visto pelo usuário como um computador. Existem também aplicações mais genéricas nas quais o computador pode ser usado para desenvolvimento de programas, aquisição de dados e outras aplicações de processamento nas quais controla um processo em tempo real.

Como exemplos de aplicações dedicadas temos:

- Controle de processos: é onde está a maior parte das aplicações de tempo real. Em muitos casos o computador é aparte de outro produto não se apresentando ao usuário como um computador de uso geral ou mesmo como se fosse um computador.
- As aplicações mais elaboradas encontram-se em laboratórios, equipamentos e plantas industriais. Os processos sendo controlados incluem reações químicas, linhas de montagem, máquinas de controle numéricos, plantas de processos contínuos (celulose, petroquímica).
- Telecomunicações: as aplicações neste campo são de muitas e variadas. Incluem centros de chaveamento de mensagens, centrais telefônicas, redes de computadores. Aplicações em menor escala encontram-se em multiplexadores e modens inteligentes.
- Instrumentos inteligentes: é cada vez maior o número de instrumentos que incorporam microcomputadores. Por exemplo, multímetros digitais com calculadora, osciloscópio digitais, analisadores de assinatura, etc...
- Robôs: esta é uma aplicação típica de controle de processos: a movimentação do robô é um processo maior de manufatura. Pelo fato de necessitarem mover-se rápida e constantemente através de trajetórias complexas os robôs industriais representam um amplo campo de aplicação para microcomputadores.

2.3 Taxinomia

Os sistemas de tempo real podem ser classificados segundo duas linhas básicas: de acordo com a sua velocidade (tempo de resposta ou desempenho) e de acordo com a estrutura de seu software.

2.3.1 Quanto a velocidade

Os sistemas de tempo real podem ser classificados quanto a velocidade em sistemas de baixa, média e alta velocidade em função dos tempos de respostas requeridos e da capacidade de processamento da máquina utilizada.

Esta classificação é detalhada a seguir sendo mencionados os tempos de resposta típicos para cada uma delas.

- Baixa velocidade: estas aplicações caracterizam-se pelos tempos de respostas em segundos ou minutos. Aplicações típicas são sistemas de aquisição de dados de baixa velocidade (por exemplo contagem de veículos em uma rodovia), e controle de processos lentos tais como sistemas de controle de demanda elétrica.
- Média velocidade: neste caso os tempos de resposta estão na ordem de milissegundos ou segundos. O software já necessita de alguns cuidados: eventos devem ter prioridades, os dados devem ser armazenados, alguns trechos críticos podem ser escritos em linguagem de baixo nível para maior rapidez de processamento. Uma característica importante aqui é que tempos de resposta começam a ficar próximos do tempo gasto pelo computador para realizar as ações.
- Alta velocidade: nestas aplicações os tempos de respostas envolvidos são aproximadamente iguais ao da capacidade do computador. Os sistemas de alta velocidade são geralmente simples pois não há tempo para a execução de algoritmos mais complexos. Os tempos de resposta vão desde dezenas de

microssegundos a dezenas de milissegundos. Como exemplos típicos temos os filtros digitais e outros processamentos de sinais.

2.3.2 Quanto a estrutura

- Polling loop: é a estrutura mais simples para sistemas de tempo real: tipicamente o programa examina (polls) cada uma das entradas para determinar em qual delas ocorreu um evento que requeira uma resposta. Caso tenha ocorrido um vento o programa executa as ações correspondentes antes de examinar a próxima entrada. Em algumas aplicações como, por exemplo, em alguns controles lógicos programáveis (CLP) o programa inicialmente examina todas as entradas, realiza as ações correspondentes e finalmente executa a saída dos dados.

A característica básica desta estrutura é que os eventos externos não interrompem o programa sendo somente detectados quando a entrada correspondente for verificada.

É interessante ressaltar que esta estrutura é encontrada tanto nos sistemas de baixa velocidade como nos de alta velocidade. Esta simplicidade na estrutura é desejável nos sistemas de baixa velocidade e necessária nos de alta.

- Sistemas orientados a eventos (event-driven) são os sistemas que respondem diretamente a um vento externo. Esta designação abriga três tipos principais de sistemas que são: foreground/background, multitarefa e multiprocessador.
- Foreground/background: está é provavelmente a estrutura mais usada. Geralmente tem-se um polling loop rodando em “background” enquanto os eventos causam interrupções no “foreground”. Quando ocorre uma interrupção o controle é transferido para uma rotina de atendimento e após seu processamento o controle é retornado para o background. Usualmente o foreground atende as interrupções e armazena os dados da estrutura que posteriormente serão acessadas pelos programas rodando em background.
- Multitarefa: um sistema multitarefa é aquele que tem seu software (aplicativo) dividido em pois ou mais programas (tarefas) independentes. As tarefas podem rodar em um mesmo processador (multiprogramação) ou em processadores

diferentes (multiprocessamento) sendo providenciado meios para suas interações (comunicações e sincronização).

- A multiprogramação é suportada por software, normalmente por um componente, denominado núcleo do sistema operacional, que tem como função fazer com que cada programa, chamado de tarefa ou processo, execute como se tivesse o seu próprio computador. Isto é feito pelo salvamento do contexto de uma tarefa (registradores, flags e outras informações) e o carregamento do contexto de outra. Existe uma grande gama de tipos de sistemas operacionais que se diferenciam pelas ferramentas para escalonamento de tarefas, sincronização de ações e troca de dados. Sistema operacionais para aplicações de tempo real representam um grande número destas facilidades.

Ao longo deste trabalho o termo “tarefa” será adotado para designar um programa de execução. O termo “processo” não será usado por apresentar um sentido mais geral que em aplicações de controle é mais adequado para indicar o sistema a ser controlado.

- Multiprocessador: uma atitude bastante comum no caso de haver muito trabalho a ser realizado pelo computador é acrescentar outro computador para auxiliar no desempenho das tarefas. Como resultado disto obtem-se um sistema multiprocessador. Pelo fato dos microprocessadores aumentam a capacidade com uma redução de custo estes sistemas estão tornando-se cada vez mais utilizados.

No caso dos processadores não possuírem memória compartilhada o sistema é classificado como processamento distribuído.

O processamento distribuído pode ser usado não somente em casos onde é necessário um aumento de capacidade de processamento. Em aplicações industriais é muito comum sua utilização para redução das conexões para transmissão de sinais a longas distâncias. Um exemplo usual é a colocação de vários computadores distribuídos por uma grande planta industrial. Estes micros podem coletar dados vindos de termopares e sensores, realizar uma consistência e conversão para unidades de engenharia e transmiti-las via uma linha serial para uma estação (computador central). Esta estação central realiza o processamento das informações e envia as decisões para os algoritmos de controle locais dos diversos microprocessadores distribuídos. Sem a utilização dos processadores distribuídos seriam necessárias longas linhas de transmissão dos sinais de sensores, expostas a todos os ruídos de um ambiente industrial.

Do ponto de vista do programador existe uma importante diferença entre multitarefa e processamento distribuído, relacionada com a comunicação entre tarefas. Em um sistema como um único processador com memória compartilhada, a existência de memória compartilhada permite uma fácil comunicação entre tarefas pelo acesso a estruturas de dados comuns às tarefas comunicantes. Em um sistema distribuído, a comunicação deve ser feita pela troca explicita de mensagens. Devido a isto devem ser tomados cuidados durante a definição do sistema com relação aos danos que devam ser compartilhados. Posteriormente, pode não ser possível uma nova distribuição das tarefas entre estações de sistemas, a menos que se disponha de ferramentas que tornem a comunicação independente da disposição das tarefas no sistema.

3 SISTEMAS DE CONTROLE DISTRIBUÍDO PARA CONTROLE DE PROCESSOS INDUSTRIAS

O controle de processos industrias por computadores já é uma realidade a algum tempo, assim como o controle digital direto (DDC), capaz de efetuar controles não lineares. O aparecimento de sistemas de controles onde processadores são distribuídos estrategicamente pela planta de acordo com o planejamento e necessidades do processo, e utilizados para efetuar processamento de cada local, permite em tese obter uma eficiência ótima associada a uma alta confiabilidade. Estes sistemas de controle onde múltiplas estações de controle baseadas em mini ou microcomputadores operam cooperativamente para realizar o controle de uma planta são denominados sistemas de controle distribuído ou sistemas de controle com inteligência distribuída / IMO 83/.

Os sistemas de controle distribuídos existentes na indústria (tais como: petroquímica, papel e celulose, aço e eletricidade) são um conjunto de funções bem definidas na forma de módulos interconectados por meio de comunicação. Necessidades tais como tempo de resposta, comunicação com o operador, throughput, flexibilidade, backup e recuperações de erros, vem estimulando o

desenvolvimento de sistema de controle de processos na forma de uma organização distribuída unificada em termos de hardware e software /SCH 84/.

3.1 Principais funções do controle de processos industriais

Os sistemas voltados para o controle de processos industriais tem como principais funções:

- Aquisição de dados: pelo fato dos processos serem dinâmicos, sinais analógicos devem ser acessados, amostrados e digitalizados, tipicamente, em velocidade de até 10 vezes por segundo. Sinais digitais tais como chaves de fim de curso, limites, etc... Também devem ser adquiridos do processo.

O sinal fornecido pelo sensor deve ser convertido para unidades de engenharias, filtrado, verificado os limites para alarme e finalmente armazenado em uma estrutura de dados, denominada base de dados, para posterior uso pelas diversas tarefas do sistema. Cada etapa deste processamento possui muitas alternativas que dependem do tipo de sensor, ruído variabilidade do sinal, distúrbios que alterem o sinal, importância do sensor para o processo, etc. Estas funções são tipicamente realizadas por diversos modelos de modo a aumentar a throughput do sistema de forma que a base de dados obtida é distribuída.

Os mecanismos de software para validação, armazenamento e recuperação das informações da base de dados dependerão da aplicação desejada bem como do tamanho da base de dados, tempo de resposta, confiabilidade, etc.

- Controle de variáveis do processo: o controle envolve variáveis medidas ou calculadas, valores de referência (set points) e algoritmos de controle que determinam variações nos valores de referência das variáveis para seu controle e otimização. Os algoritmos de controle variam consideravelmente na sua complexidade pois dependem das características dinâmicas dos processos, tais como realimentação, controle antecipatório (feedforward), cascata e sequências de coordenação e controle.

Os laços de controle operam periodicamente, geralmente algumas vezes por segundo por laço, e algumas vezes utilizando dados de módulos remotos.

- Comunicação com o operador: em todos os sistemas existe a necessidade de comunicação com o usuário, a qual inclui a apresentação de informações de base de dados. Além disto, o sistema deve ter condições de modificar operações alterando valores de referência de variáveis, modificando parâmetros dos algoritmos, alterando configurações, etc.

Os módulos de comunicação com o operador geralmente consiste em um ou mais terminais (usualmente com vídeos coloridos), com teclados dedicados, um ou mais processadores e uma memória secundária local. Pelo fato da base de dados estar distribuída pelos diversos módulos, são feitas cópias, constantemente atualizadas, para visualização, modificação e mesmo para partida do sistema após paradas. Esta comunicação permite ao sistema operar sem um computador supervisor.

- Sistema supervisor: os modernos sistemas de controle de processos são organizados em uma arquitetura hierárquica onde as funções críticas de tempo estão no nível superior e funções menos críticas com relação ao tempo, mas mais complexas, nos níveis superiores.

Aquisição de dados e o controle digital direto constituem o primeiro nível de controle. As aplicações do nível superior geralmente incluem otimização de desempenho, gerência de energia, gerência da planta completa, e planejamento de produção. Estas funções de controle geralmente residem em um computador de uso geral interfaceado com o restante do sistema de modo que a base de dados fique disponível para as tarefas encarregadas de executar as funções de nível superior. Estas aplicações variam de caso a acaso dependendo dos objetivos a serem atingidos e do tipo da indústria. Consequentemente este nível geralmente é programado pelo usuário usando alguma linguagem de alto nível convencional e um sistema operacional em tempo real.

4. UM MODELO PARA SISTEMAS DE CONTROLE DISTRIBUÍDOS

Um sistema distribuído é composto por diversos elementos distintos que trabalham cooperativamente. Cada componente do sistema deve ser autônomo de forma a gerenciar e controlar seus próprios recursos. Quanto ao usuário, este deve enxergar o sistema como um todo onde a distribuição dos recursos é transparente.

Um ponto importante na construção de sistema distribuído é a modelagem. A escolha de um modelo de arquitetura fornece subsídios que facilitam a construção do sistema.

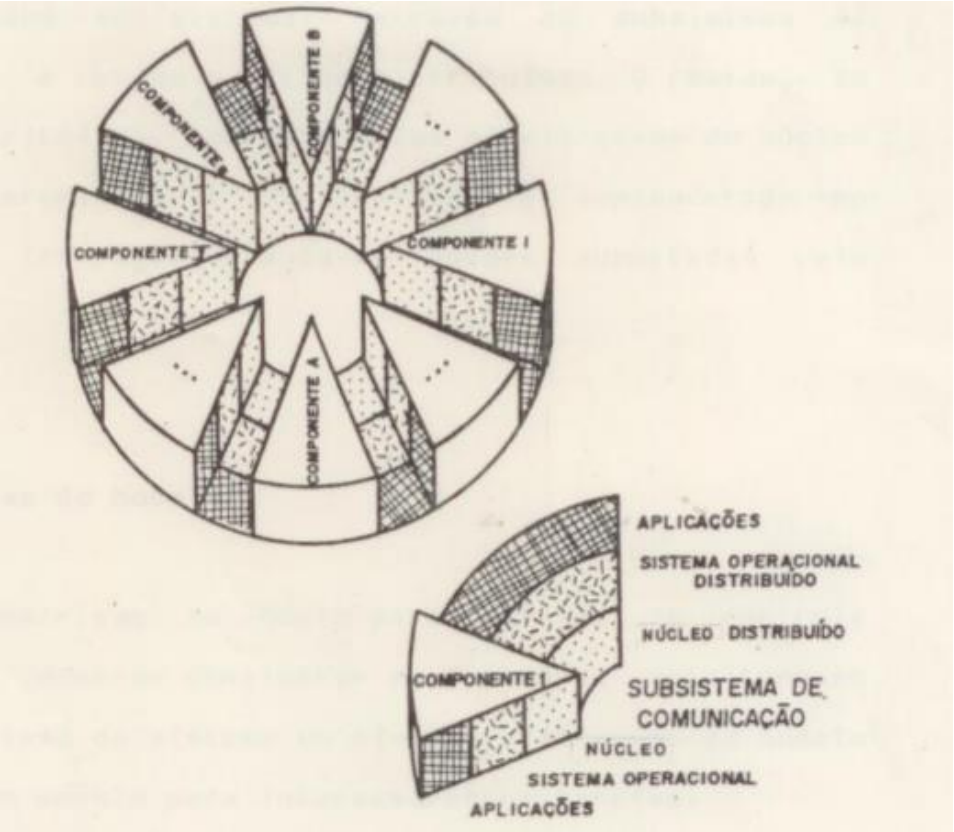
A base para implantação do modelo de sistema distribuído é formada por um conjunto de sistemas de computadores autônomos (estações ou nodos) interconectados através de um subsistema de comunicação /WAT 80/

O modelo adotado é baseado no proposto por Queiroz para construção de sistemas distribuídos /QUE 84/.

A arquitetura do modelo é apresentada em níveis sendo especificados os serviços fornecidos por cada nível bem como os serviços requeridos pelos mesmos.

Esta arquitetura é fortemente apoiada em um sistema operacional que fornece os mecanismos para obter um sistema distribuído.

Figura 1 - Arquitetura do modelo para Sistemas de Controle Distribuídos



Fonte: O autor, 1987.

Além do nível correspondente ao subsistema de comunicação o modelo compreende três outros níveis, os quais são: núcleo, sistema operacional e aplicações. O núcleo de cada nodo ou estação realiza o controle e gerencia local, oferecendo entre outras facilidades a sincronização e a comunicação entre tarefas. O núcleo também realiza a interface com o subsistema de comunicação, é obtido o núcleo distribuído. O restante do sistema distribuído, composto pelos níveis acima do núcleo (sistema operacional e aplicativos) é implementado por tarefas que interagem através de funções suportadas pelo núcleo.

4.1. Estruturas do modelo

Para descrição do modelo para sistemas de controle distribuídos deve-se considerar as formas de estruturação que são: divisão do sistema em níveis, utilização do modelo de objeto e o modelo para interação entre tarefas.

4.1.1 Arquitetura em níveis

Este é um conceito bastante utilizando na Ciência da Computação para definição da arquitetura de sistemas. Desta forma um estrutura complexa pode ser dividida em diversos níveis funcionais independentes.

Cada nível possui um conjunto bem definidos de funções que utilizando-se do nível inferior fornece serviços para o nível superior. A comunicação entre os dois níveis adjacentes é realizada através de uma interface bem definida.

4.1.2 O modelo de objeto

O modelo de objeto /JON 78/ é tanto um conceito como uma ferramenta para o projeto de sistemas de software sendo utilizada para representação das diversas entidades componentes. Como este modelo é possível a caracterização das entidades abstratas a serem modeladas por um sistema computacional.

Objetos são os componentes passivos do sistema sendo, justamente com o seu comportamento, caracterizados por propriedades invariantes. Estas propriedades são preservadas por um conjunto de operações que fornecem o único meio de um objeto ser manipulado. Objetos com as mesmas característica de comportamento podem ser definidos por um único conjunto de operações. Dois objetos são do mesmo tipo quando compartilham um mesmo conjunto de operações.

Um módulo de tipo agrupa as operações de tipos para sua implantação. Um objeto só pode ser diretamente manipulado através das operações de seu tipo. Um novo tipo pode ser definido através de tipos já existentes. Desta mesma maneira, objetos novos podem ser criados. Um sistema pode ser implantado completamente pela construção de um conjunto de módulos de tipo relacionados por dependência (quando as operações de um módulo do tipo são implementadas através das operações de outro).

Sistemas aplicativos ou sistemas operacionais podem ser construídos aplicando o modelo de objeto. Em se tratando de sistemas operacionais uma das principais metas de projeto é o acesso aos objetos. Os objetos podem ser reais, como: memória, processador, dispositivos de E/S; ou abstratos tais como: arquivos, diretórios, tarefas. Estes objetos fornecem os meios básicos para construção de outros objetos. Um objeto é especificado por um conjunto de operações que podem ser executadas sobre a representação. A utilização de recursos pode ser feita apenas através da invocação de operações ou funções como parâmetros associados.

O modelo de objeto está fundamentado na noção de domínio, devendo ser suportado por mecanismos que suportem esta noção.

O suporte ao domínio deve permitir a troca do domínio de execução com o possível retorno. Deve ser permitida a suspensão da execução de um determinado domínio para permitir a execução de outro, sendo possível o retorno ao domínio inicial que foi suspenso.

4.1.3 O Modelo para interação entre tarefas

Apesar de independentes as tarefas de um sistema distribuído devem executar cooperativismo de modo a atingir seus objetivos. Os objetivos que compõem o sistema se comunicam através de sua tarefas. A comunicação entre tarefas pode ser realizadas de modo síncrono ou assíncrono.

Na comunicação assíncrona, uma tarefa que que deseje se comunicar com outra, o faz de maneira independente desta outra tarefa, sem sincronização. A comunicação é feita pelo despacho das informações para a outra tarefa. Para

comunicação não é necessário que tarefa de origem (ou transmissora) tenha sua execução interrompida, podendo continuar o seu fluxo normal de execução. As informações enviadas serão resgatadas pela tarefa destino (ou receptora) a qualquer tempo e independente do estado da tarefa que iniciou a comunicação. Já na comunicação síncrona ocorrem esperas potenciais pois a comunicação entre duas tarefas depende do estado de cada uma delas. Neste caso a tarefa que iniciou a comunicação fica parada a espera de que a outra se comunique com ela para que haja a troca de informações. Após a efetivação da comunicação, ambas as tarefas seguem executando a partir do ponto onde pararam / AND 83/.

Na sua maioria, os sistemas convergem para um dos dois principais modelos de interações (comunicação e sincronização) entre tarefas, que são o orientado a mensagens e o orientado a procedimentos /LAU 78/. A escolha de um ou outro modelo depende da filosofia de relacionamento entre as tarefas /AND 81/ e da arquitetura do hardware onde o sistema será implementado.

O modelo orientado à mensagens caracteriza-se por apresentar facilidades para a troca de mensagens entre as tarefas. Neste modelo as estruturas de dados manipuladas pelas tarefas que estão interagindo são passadas (by reference) em mensagens. No modelo orientado a procedimentos a procedimentos as tarefas interagem por meio de mecanismos que permitem o endereçamento e a proteção às estruturas de dados comuns; o que implica na obrigatoriedade da existência de uma área de memória compartilhada. A sincronização usualmente é feita através de estruturas do tipo semáforo /DIJ 72/, monitores /HOA 74/ ou região critica / HAN 73/.

Como as tarefas são independentes, os sistemas orientados a mensagens são adequados tanto para o uso em arquiteturas distribuídas como para arquiteturas de memória compartilhada.

Por este fato foi adotado o modelo de comunicação e sincronização entre tarefas por troca de mensagens.

4.2 Os níveis de componentes do modelo

A seguir são feitas algumas considerações sobre os níveis da arquitetura proposta.

4.2.1 O nível de aplicação

Este é o nível superior da arquitetura onde estão localizadas as tarefas que constituem os aplicativos. Os serviços fornecidos por este nível devem ser orientados a aplicação a qual o sistema se destina. Este nível determina os serviços a serem fornecidos pelo sistema operacional.

Este campo de aplicação distribuídas ainda não está muito desenvolvido sendo objeto de diversas pesquisas. Duas áreas importantes no desenvolvimento de aplicação distribuídas são as estruturas de aplicação e as linguagens, por apresentarem problemas fundamentais a serem resolvidos /WAT 80/. Com relação a estrutura de aplicação os problemas envolvem a organização e estruturação, tanto física como logica, do processamento dos dados e outros recursos de aplicação /ENS 78/. Com relação as linguagens, é necessário que se desenvolvam linguagens para interfaceamento com o usuário em seu terminal e com o sistema operacional, e para programação de sistemas distribuídos.

Neste aspecto de linguagens de programação o campo relacionado a aplicações de tempo real em controle de processos apresenta um estado de incipiência nem maior do que as restantes aplicações de sistemas distribuídos (automação de escritórios, automação bancária, correio eletrônico e outros) apresentando um amplo campo para desenvolvimento.

4.2.2 O nível de sistema operacional distribuído

O nível de sistema operacional distribuído é fornecido pela conjunção de todos os serviços fornecidos pelos diversos servidores, que se encontram distribuído pelo sistema.

Estes serviços são fortemente influenciados pela aplicação. Esta área de desenvolvimento, está de maneira geral, melhor entendida do que as aplicações. Prova disto são os diversos sistemas operacionais distribuído em desenvolvimento ou já operacionais /CHE 84/ /LEB 84/.

Do ponto de vista da aplicação os serviços deste nível são em sua maioria semelhantes aos prestados por sistemas operacionais não distribuídos porque os tipos básicos de objetos fornecidos são os mesmos. As principais diferenças introduzidas pelos sistemas operacionais distribuídos estão relacionados com sua organização interna e implantação.

4.2.3 O nível de núcleo distribuído

O núcleo distribuído é formado pela conjunção de todos os núcleos que interagem através de subsistemas de comunicação. E através do nível de núcleo distribuído que as tarefas realizam a comunicação e sincronização. São também funções deste nível fornecer meios para proteção do sistema, multiplexação de recursos, criação e gerência de tarefas, e operações de E/S entre outros. A interação entre tarefas pode ser local (tarefas de um mesmo nodo) ou remota (tarefas residentes em nodos distintos). De qualquer forma os mecanismos fornecidos pelo núcleo devem ser tais que as tarefas interajam do mesmo modo, independentemente da localização das mesmas e independente de serem estas tarefas fornecidas pelo sistema ou pelo usuário.

As tarefas interajam com o núcleo através de chamadas ao sistema ou TRAPS. Ao executar um trap a tarefa solicita o fornecimento de um serviço de núcleo. O núcleo executa de modo não interruptível (interrupções desabilitadas) sendo o único componente do software que toma conhecimento da ocorrência de interrupções. No caso da ocorrência de uma interrupção quando uma tarefa esteja sendo executada esta perde o processador que segue executando a rotina do núcleo encarregada do atendimento da interrupção. Após a interrupção ser atendida

o processador é entregue à tarefa de maior prioridade que segue executando. Os serviços fornecidos por cada núcleo componente variam, dependendo das características do hardware, segurança e aplicações a serem suportadas. Cada núcleo componente oferece apenas os serviços suportados pelo nodo ao qual pertence.

4.2.4 O subsistema de comunicação

O subsistema de comunicação suporta a troca de mensagens entre as diversas estações componentes do sistema distribuído. Ele fornece a interface pela qual as tarefas tem acesso as facilidades implementadas pelos protocolos de comunicação. E neste componente do software de uma estação que são implantados os protocolos de comunicação.

O tratamento recebido pelas mensagens depende dos protocolos utilizados e do modelo adotado. Entretanto independente do modelo escolhido, o subsistema de comunicação deve fornecer um serviço seguro de troca de mensagens o que se traduz, de um modo geral, em suportar o roteamento das mensagens, garantir a correção das mesmas, a sequência correta de envio e controle de fluxo entre outras funções.

Em qualquer caso, a comunicação entre duas estações se dará através de uma interface bem definida, visando tornar cada núcleo componente de uma estação totalmente independente do modelo e protocolos utilizados no subsistema de comunicação.

Isto foi adotado de modo a não impor nenhuma restrição em relação ao subsistema de comunicação, podendo ser usado tanto uma rede local com níveis de protocolos, topologia e meios de comunicação variados, como um sistema canal de comunicação serial e protocolos simplificados. Isto é justificável pois dependendo do número de estações envolvidas, distância entre as mesma e trafego de mensagens nem sempre é justificável a utilização de uma rede local, mesmo em aplicações industriais intensivas /CAN 85/.

Um outro fator a ser considerado também, é que o desenvolvimento dos meios de comunicação e a sua visibilidade de implementação pelos fabricantes, foi fortemente influenciado pela capacidade de processamento e pelo nível de inteligência disponível em cada elemento do sistema distribuído e do seu custo na época. Isto impeliu a que cada fabricante adotasse soluções próprias que mais se adequassem a seus sistemas específicos, levando ainda a que muitos considerem os métodos de acesso e os dados que trafegam na via expressa como propriedade industrial / IMO 83/.

Estes fatos resultam em uma dificuldade na obtenção de um consenso com relação a um modelo padrão para conexão de sistemas de controle distribuídos /ALL 86/ /DOO 85/.

NA realidade este problema é bem mais amplo pois não abrange somente a interconexão de sistemas de controle como também todas as aplicações que necessitem a interconexão de sistemas de fabricantes distintos / FOL 85/ /STA 84/.

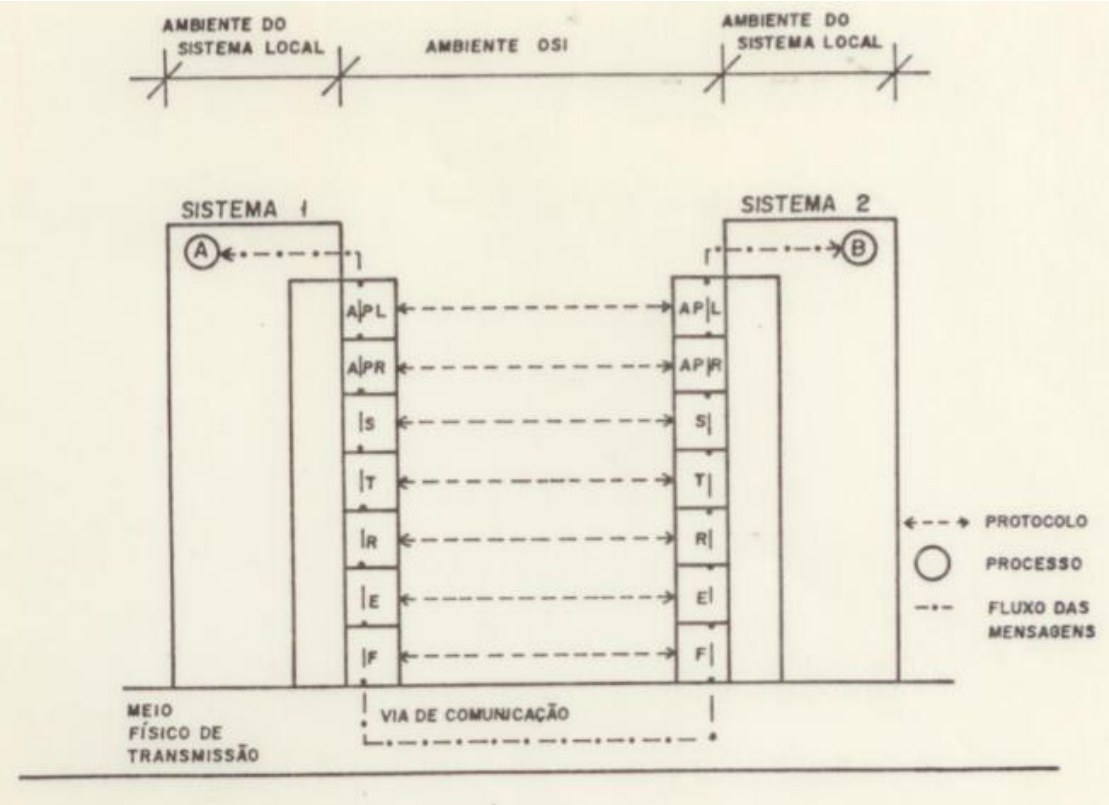
Várias tentativas vem sendo feitas por diversos organismos visando obter um modelo padrão para interconexão. Surge aí o modelo básico de referência da ISO para interconexão de sistemas abertos (OSI) /INT 80/. O termo “aberto” indica que um sistema de um fabricante pode ser conectado a qualquer outro sistema desde que ambos estejam de acordo com o modelo de referência e tenham implantados os protocolos associados.

4.2.4.1. O modelo básico de referência da ISO para interconexão de sistemas abertos (OSI)

O objetivo do modelo de referência OSI é permitir a interconexão e comunicação entre aplicações de usuários espalhados por um largo espectro de sistemas, sem que seja necessário que o usuário se preocupe com as diferenças dos sistemas envolvidos na interação. A estrutura lógica do modelo é universalmente aplicável e pode ser usada para analisar áreas onde padrões são necessários, para determinar a adequação de padrões existentes e tornar possível a evolução ordenada da tecnologia /FOL 81/.

O modelo foi definido tendo como base a partição ou agrupamento lógico das funções necessárias em sete níveis. Cada nível é composto por unidades separadas funcionalmente que fornecem um conjunto especializados de serviços.

Figura 2. O modelo de referência ISO



Fonte: autor, 1987.

A figura 2. Apresenta o modelo de referência e os seus sete níveis. Componentes, os quais são descritos a seguir.

- Nível de aplicação (7): é o nível mais alto da arquitetura OSI, sendo o único que interage com o usuário final. Fornece e gerencia todos serviços que possam ser diretamente usados pelos usuários (tarefas).
- Nível de apresentação (6): este nível está relacionado com as características sintáticas dos dados. Seus serviços incluem transformações de dados (convenções de produtos de caracteres e de códigos), formatação de informação (modificação do lay-out dos dados); e seleção de sintaxe (seleção inicial e modificação subsequente das transformações e formatos usados).
- Nível de sessão (5): é o último dos considerados altos níveis da arquitetura OSI. Este nível como os anteriores, não se envolve diretamente com os mecanismos de ligação em redes, sendo a interface do usuário com esta. Sua função é coordenar a interação entre os níveis de apresentação pelo

estabelecimento de um caminho lógico para transferência de dados. Seus serviços incluem: o estabelecimento e liberação de uma conexão de sessão, o gerenciamento de uma conexão de sessão; troca de dados do usuário; envio de dados de quarentena; e gerenciamento da interação.

- Nível de transporte (4): o nível de transporte garante a integridade da comunicação fim-a-fim e a qualidade necessária para o serviço de troca de informações.

Entre os serviços fornecidos por este nível estão: estabelecimento de conexão de transporte entre entidades de sessão; seleção de classe de serviço; transparência de dados; e encerramento da conexão de transporte.

- Nível de rede (3): Prove os meios para que sejam estabelecidas, mantidas e encerradas as conexões. Ele fornece conexões de rede para que unidades de dados sejam trocadas entre unidades de transporte.

Os serviços fornecido para este nível são, entre outros: estabelecimento, manutenção e encerramento de uma conexão de rede, notificação de erro; controle de sequência de unidades de dados; e confirmação de entrega de unidades de dados.

- Nível de enlace (2): Fornece os meios funcionais e procedurais para estabelecer, manter e encerrar uma conexão de enlace entre entidades de rede.

Seu objetivo é detectar e possivelmente corrigir erros ocorridos no nível físico.

Dentre os serviços fornecidos estão: fornecimento de conexão de enlace; trocas de unidades de dados; notificação de erro; controle de sequência de unidades de dados; e controle de fluxo.

- Nível de físico (1): este é o nível inferior da arquitetura OSI. Este nível fornece os meios funcionais e procedurais necessários para ativar, manter e desativar conexões físicas para transmissão de bits entre entidades de enlace, utilizando diretamente o hardware disponível e levando em conta as características físicas do meio de comunicação.

Os serviços fornecidos incluem: conexão física; transmissão de unidade de dados; sequenciamento de entrega; e notificação de condições de falha.

4.2.4.2. Relacionando o modelo de referência ISO com o modelo de sistema distribuído

É interessante analisar o relacionamento entre o modelo de referência ISO e a arquitetura proposta para o sistema de controle distribuído.

O modelo de referência destina-se a interconexão de sistemas autônomos e por ser uma norma necessita definir uma estrutura sofisticada e complexa de protocolos visando um maior dinamismo na interconexão de aplicações. Esta estrutura é definida através de sete níveis. Os primeiros quatro níveis (físico, enlace, rede e transporte), são denominados baixo nível e estruturam os protocolos encarregados da comunicação propriamente dita. Os três níveis restantes (sessão, apresentação e aplicação), que constituem o protocolos alto nível, tem suas funções voltadas para o processamento de informações.

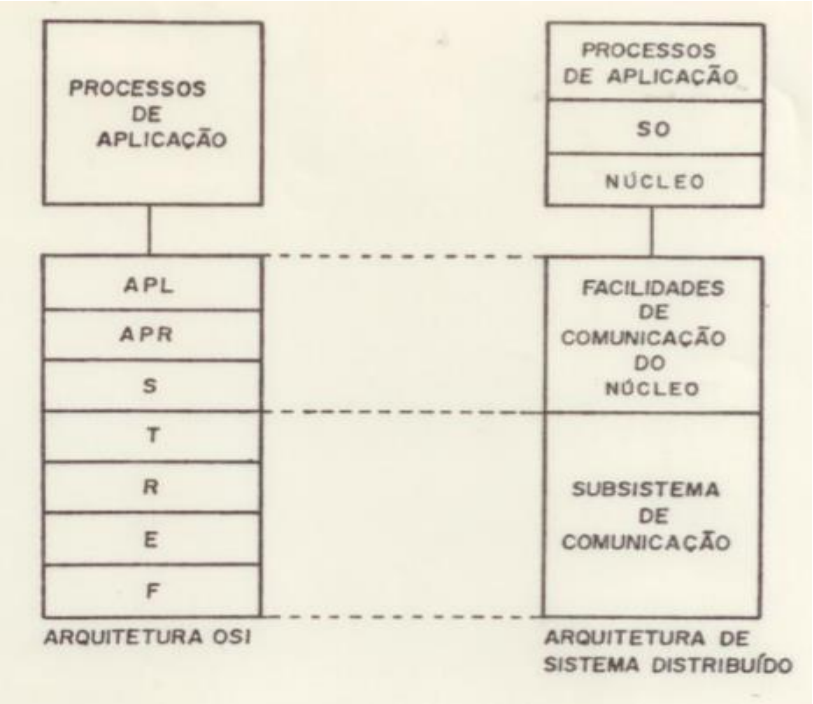
O modelo de sistema distribuído vai além da estrutura dos protocolos de comunicação pois estrutura todo o sistema e não somente subsistema de comunicação. Seu objetivo é fornecer todos os serviços necessários para uma aplicação distribuída, onde o usuário enxerga o sistema como um todo, sendo transparente a distribuição dos recursos.

Os protocolos necessários ao subsistema de comunicação pode se tratar de uma aplicação especifica, não necessitam da complexidade e sofisticação dos apresentados no modelo de referência ISO. Entretanto é interessante que sejam estruturados de modo a seguir as recomendações do modelo de referência.

Numa comparação entre os dois modelos pode-se identificar que o nível de subsistema de comunicação tem funções semelhantes aos protocolos de baixo nível. Desta forma os quatro níveis inferiores do OSI da ISO constituíram o subsistema de comunicação sendo os três níveis restantes (protocolos de alto nível) implementados pelos mecanismos de comunicação de tarefas do nível de núcleo distribuído. As demais funções do núcleo, o sistema operacional e aplicações podem ser considerados como o nível de aplicações do modelo ISO.

A figura 3. Mostra uma possível forma de relacionamento dos modelos.

Figura 3. As arquiteturas OSI e sistema distribuído



Fonte: Autor, 1987.

5. A IMPLEMENTAÇÃO DE UM AMBIENTE PARA CONSTRUÇÃO DE SISTEMA DE CONTROLE DISTRIBUÍDOS

O grande desenvolvimento apresentado pelos microcomputadores propiciou sua utilização para solução de um grande número de problemas na área de controle de processos industriais. Entretanto tais aplicações encontram como principal obstáculo a inexistência de um ambiente computacional que permita uma eficiente manipulação dos recursos suportados pelo hardware aliado à ferramenta de software que permitam incorporar o estado da arte no tocante a estruturação, definição e implementação de tais sistemas.

Como decorrência dos trabalhos desenvolvidos pelo Grupo de Controle de Processos, em especial no projetos PROCON, verificou-se a necessidade de obtenção de um ambiente computacional adequado a construção de software para sistemas de controle de processos, notadamente em aplicações utilizando diversos microcomputadores interconectados.

Esta constatação motivou a decisão de obtenção de um sistema que apresentasse o ambiente com as características desejáveis para o desenvolvimento de aplicações e de pesquisas em sistemas de controle distribuído utilizando microcomputadores.

O sistema desejado deve apresentar, em cada estação componente de um sistema distribuído, um ambiente computacional com as seguintes características principais:

- Utilização de uma linguagem de alto nível;
- Ser multiprogramado e oferecer facilidades para aplicações de tempo real;
- Comunicação e sincronização entre tarefas realizadas de maneira uniforme e independente da distribuição das tarefas dentro do sistema;
- Recursos para manipulação de eventos;
- Independência do meio de conexão entre estações;
- Modularidade de modo a permitir implementação e validação graduais das características acima mencionadas.

Devido a indisponibilidade de um ambiente com estas características, optou-se pela construção do mesmo, utilizando-se uma linguagem sequencial cujo compilador permita a inclusão das rotinas externas (escritas assembly) /CAN 84/.

Esta solução permite incorporar ao sistema original as características desejadas, representando um compromisso, entre a utilização de linguagens orientadas para este fim /AND 81//HAN 78//HOR 78//KRA 84/, atualmente pouco difundidas e em geral complexas, e a solução através de artifícios que simulam a nível de aplicativos o ambiente desejado, o que torna a programação lenta e pouco flexível.

5.1. Estrutura do sistema

A estrutura adotada foi aquela apresentada no modelo para sistemas de controle distribuídos (capítulo 4).

Desta forma tem-se que o software de uma estação é dividido em quatro níveis componentes:

a. Nível de aplicação;
b. Nível de sistema operacional;
c. Nível do núcleo;
d. Nível do subsistema de comunicação.

No sistema implementação, pela decisão de se utilizar uma linguagem sequencial convencional, o nível de aplicação é suportado pelo compilador da linguagem escolhida, o nível de sistema operacional é constituído pelo sistema operacional nativo sobre o qual o compilador escolhido roda. Os níveis restantes (o de núcleo e o de subsistema de comunicação) deverão ser desenvolvidos e acrescentados ao software de um microcomputador convencional de modo a transformá-lo em uma estação do sistema distribuído.

Desta forma as características adicionais necessárias as aplicações em questões serão definidas a nível de aplicativo como rotinas externas (escritas assembly). Tais rotinas serão partes integrantes do núcleo da estação.

O ambiente básico (sistema operacional original e utilitários) é utilizado para construções de aplicativos. Para execução dos mesmos é feita a ligação e carga dos diversos módulos que compõem o software da estação (aplicativo + tarefas do sistema + sistema operacional + núcleo + subsistema de comunicação).

A seguir serão apresentadas as características desejáveis de uma linguagem para aplicação de tempo real. No capítulo seguinte é descrito o núcleo de estação, sua estrutura e funções principais.

5.2 Características de uma linguagem para aplicações de tempo real

Uma linguagem de programação de alto nível orientada para aplicações de tempo real deve apresentar algumas características e propriedades as quais são apresentadas a seguir /YOU 78/:

a) Portabilidade: idealmente a linguagem deve ser independente da máquina. Na prática, isto não é facilmente obtido devido as diferenças no tamanho das palavras e nos mecanismos de E/S.

b) Segurança: as construções da linguagem devem ser concebidas de forma que permitam a detecção do maior número possível de erros em tempo de compilação e durante a execução.

c) Estrutura: deve prover estruturas de dados adequadas para construções de sistemas de tempo real. Em particular deve prover tipos de dados tais como bits, bytes, matrizes, registros e apontadores.

A estrutura dos comandos deve ser tal a permitir que e a codificação em si seja auto documentada e explanatória de modo a estimular o uso dos princípios de programação estruturada.

d) Eficiência: o código objeto gerado deve ser compacto e eficiente.

e) Simplicidade: as construções da linguagem devem ser simples e consistentes de modo a permitir um fácil aprendizado pelos usuários além de facilitar a construção de compiladores.

f) Flexibilidade: a linguagem deve ser adequada para todos os aspectos do desenvolvimento de software de tempo real tanto a nível do sistema quanto a nível de usuário.

g) Concorrência: a linguagem deve suportar programação concorrente com a definição de módulos independentes.

h) Reentrância e recursividade: os procedimentos devem se reentráveis para permitir o compartilhamento de rotina por tarefas concorrentes e o uso de técnicas recursivas.

Muitas das características apresentadas acima são conflitantes. Por exemplo, a previsão de segurança em tempo de execução (b) inevitavelmente implica em uma redução substancial na eficiência contrariando (d). Na prática, o projeto de linguagem de tempo real é um compromisso entre essas necessidades conflitantes.

5.3 Linguagens de alto nível com extensões de tempo real

Linguagens de alto nível tais como BASIC, PASCAL, C e FORTRAN podem ser usadas para aplicações de tempo real, desde que sejam estendidas através de rotinas apropriadas. Normalmente é necessário utilizar rotinas assembly na programação de certas funções.

Esta abordagem tem como principal vantagem o fato de tais linguagens serem bem difundidas, fáceis de usar, além de possuírem diversas implementações (compiladores) para os mais variados tipos de sistemas.

É interessante fazer uma rápida análise das vantagens e desvantagens no uso de cada uma das quatro linguagens mencionadas.

O BASIC por ser uma linguagem interpretada, usualmente é lenta para aplicações críticas com relação ao tempo. Por outro lado em casos onde não existem restrições quanto ao tempo de execução uma linguagem interpretada pode ser interessante devido principalmente as facilidades de depuração e teste.

FORTRAN por sua vez pode ser usado em aplicações críticas de tempo com a vantagem de possuir uma grande quantidade de software de suporte na forma de biblioteca de sub-rotinas, otimizadores e depuradores. Entretanto o mecanismo de alocação estática de memória do FORTRAN, que impede a chamada reentrante de procedimentos, sua inabilidade na manipulação de endereços, característica importante para manipulação de listas, e seus tipos de dados fixos vão de encontro aos aspectos de programação de sistemas para aplicações de tempo real.

Atualmente Pascal e C são duas linguagens que estão sendo cada vez mais usadas para aplicações de tempo real. Uma comparação entre estas duas linguagens deve ser feita separadamente da avaliação de compiladores em particular. Entretanto o conceito de que o C por suas características deva permitir uma geração de códigos mais eficiente deve ser questionado. O estilo de programação possível em C, permitindo construções dependentes da máquina e o uso de efeitos colaterais, podem causar problemas para os algoritmos de otimização de código /HOD 84/.

Para construções de programas autodocumentados o Pascal é preferível ao C por possuir um conjunto mais rico em tipos de dados e construções de controle mais elegantes. A combinação da clareza da estrutura da programação do Pascal com os tipos de dados definidos pelo usuário resultam num estilo de programação que em muitas ocasiões é usado como linguagem de descrição para especificações. Um exemplo é o documento de definição da Ethernet publicado pela Xerox, Intel e DEC /XER 80/.

A popularidade do C é devida em parte a sua relação como sistema Unix e por duas razões que podem ser questionados, as quais são: permissividade e portabilidade.

A permissividade do C com suas convenções implicadas dos tipos de dados básicos e operações com bits dificultam a detecção de erros em tempo de compilação. Por sua vez o Pascal com a sua verificação de tipos (typing cheking) e sua lógica severa permite uma maior verificação de erro.

A portabilidade do C é questionável. A possibilidade de rodar um programa de um sistema em outro sem realizar alterações só é possível se tal programa for escrito seguindo um padrão bem definido. Geralmente são necessários alterações dependentes de características do compilador, da máquina ou do ambiente. Pelo fato de não possuir um padrão, como o Pascal, fica difícil determinar as

dependências de um programa escrito em C com relação aos compiladores. A dependência com relação ao hardware ocorre também mais extensivamente no C devido a sua maior permissividade. Finalmente tem-se as dependências do ambiente que ocorrem nas áreas de interface, tais como manipulação de E/S e de arquivos. Em C as rotinas de E/S não integram o escopo da linguagem sendo uma área que necessita de consenso com relação a portabilidade.

A clareza de Pascal aliada a programação modular e uso isolado de extensões da linguagem permitem a obtenção de programas altamente portáveis.

Uma comparação mais completa entre PASCAL e C é apresentada por Feuer e Gehani /FEU 82/ e Hemingway /HEM 84/. Por estas razões o PASCAL parece ser a linguagem, dentre as mencionadas, mais indicada para aplicações de tempo real.

6. NÚCLEO

O núcleo, ou kernel como as vezes é chamado, é o único componente de software do sistema executado de modo não interrompível.

O acesso ao núcleo ocorre devido a um TRAP ou a uma interrupção, sendo que o mecanismo de acesso garante que ao executá-lo o processador esteja com seu sistema de interrupções desabilitado. Por ser o único componente do sistema que toma conhecimento da ocorrência de interrupções o núcleo deve conter, obrigatoriamente, as rotinas de atendimento destas.

Externamente ao núcleo de todas as interrupções são transparentes não afetando o desenvolvimento lógico das tarefas. Uma tarefa em execução pode ser interrompida, perdendo temporariamente o processador. Este porém é o único efeito que uma interrupção causa em uma tarefa.

O núcleo cria a impressão de que cada tarefa possui seu próprio processador. Cada um destes processadores virtuais comporta-se exatamente como um processador real, exceto por ter uma taxa de execução variável, a qual é determinada pelo núcleo.

Por serem executadas de forma não interrompível as rotinas componentes do núcleo devem ser tão pequenas e simples quanto possível. A escolha destas rotinas depende do tipo de aplicação a que o sistema se destina. Uma boa política é incluir no núcleo todas as rotinas relacionadas com a segurança do sistema deixando as funções restantes serem suportadas por tarefas do sistema.

No sistema que está sendo aqui descrito as funções suportadas pelo núcleo são: funções para criação e gerência de tarefas; funções para comunicação e sincronização de tarefas; funções para controle de tempo e funções para manipulação de interrupções.

6.1 Criação e gerência de tarefas

Uma tarefa, é uma entidade de execução independente pode interagir com outras tarefas através de operações.

As tarefas executam, pelo menos conceitualmente de maneira concorrente e tem acesso exclusivo e suas variáveis. Uma tarefa apresenta ainda como propriedade o fato de seu resultado independer de sua velocidade de execução, e no caso de ser executada novamente com os mesmos dados, passar exatamente pela mesma sequência de estados fornecendo o mesmo resultado /AND 83/.

Cada tarefa é representada no núcleo por um registro descritor, denominado Task Control Block (TCB). O registro descritor (TCB), contém todas as informações necessárias para colocar as tarefas em execução. Estas informações incluem:

- O estado corrente da tarefa;
- O identificador (nome) da tarefa;
- A prioridade;
- Ponteiros para áreas de memórias alocadas para a tarefa;
- Área para salvamento dos registradores e outras informações necessárias para repor o contexto da tarefa.

É através do TCB que o sistema mantém todas informações necessárias para realizar a multiplexação do processador entre as diversas tarefas que concorrem por ele, criando a ilusão de que cada tarefa possui seu próprio processador (multiprogramação).

Durante a inicialização do sistema as tarefas são criadas através da alocação de um descritor para cada uma e sua inicialização com o contexto da mesma. A seguir estes descritores (consequentemente as tarefas), são colocados na fila de tarefas aptas (READY). A fila de aptas é organizada de acordo com a prioridade das tarefas. Assim sendo as tarefas com maior prioridade devem ficar nos primeiros lugares da fila. Entre as tarefas de mesma prioridade as que entrarem primeiro ficam na frente das demais (FIFO). Desta forma a tarefa escalonada para receber o processador será sempre a de maior prioridade e que entrou primeiro na fila.

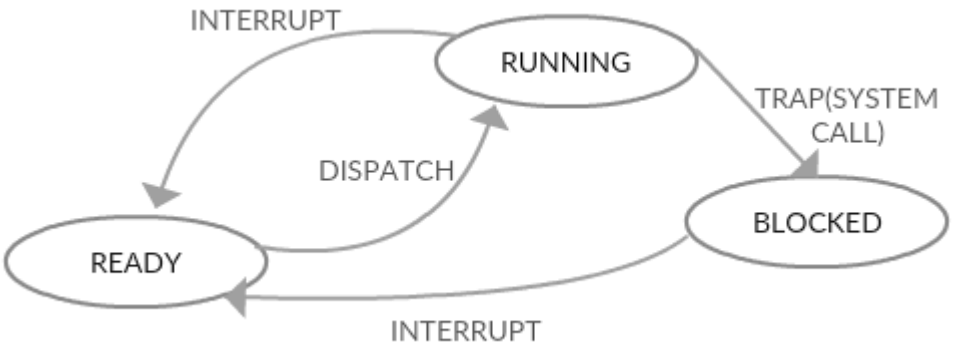
Durante a sua existência uma tarefa passa por diversos estados. De um modo geral pode-se dizer que estes estados são três:

a) “Running” – quando está usando o processador;
b) “Ready” – está pronta para receber o processador, mas este está alocado a outra tarefa;

c) “Blocked” – quando espera o término de algum serviço solicitado.

Com o escalonamento, uma tarefa é retirada da fila de aptas (READY) e seu contexto é reparado. Uma vez isto feito esta tarefa permanece com controle do processador até que seja interrompida ou solicite algum serviço do núcleo (TRAP). No caso de um TRAP o contexto da tarefa é salvo e esta é colocada na lista de tarefas bloqueadas (BLOCKED) a espera do atendimento do serviço solicitado. Uma vez terminado o serviço a tarefa que estava bloqueada é retirada da fila de espera e colocada na fila de tarefas (READY). Se a tarefa sofre uma interrupção, seu contexto é salvo no descritor e ela é colocada na READY – figura 4.

Figura 4. Diagrama de transição dos estados das tarefas.



Fonte: autor, 1987.

Com o fornecimento de mecanismos que permitam as tarefas exercerem controle sobre as outras, novos estados são acrescentados aos acima mencionados.

Geralmente a solicitação de um serviço (TRAP) faz com que uma tarefa seja bloqueada. Entretanto podem ocorrer casos (não mostrados no diagrama) em que a natureza do serviço faça com que a tarefa não perca o processador.

As diretivas para criação e gerência de tarefas são:

- START (proc. id, priorid, endepro): inicializa a tarefa com uma determinada prioridade.

proc. id: identificador da tarefa.

priorid: prioridade da mesma.

endepro: endereço inicial da área de códigos.

- QUIT: indica para o Kernel que a tarefa terminou sua execução.

6.2 Comunicação e sincronização de tarefas

Apesar de independentes, as tarefas fazem parte de um objetivo maior, devendo portanto, executarem de modo cooperativo. De forma a possibilitar esta cooperação são necessários mecanismos que permitam a comunicação e a sincronização das tarefas. A comunicação permite que a execução de uma tarefa influencie na execução de outra. Esta intercomunicação entre tarefas pode ser baseada em variáveis compartilhadas (variáveis que podem ser acessadas por mais de uma tarefa) ou por troca de mensagens.

Como a velocidade de execução das tarefas não podem ser previstas, geralmente é necessário sincronizar a comunicação. Para haver comunicação uma tarefa precisa realizar alguma ação que a outra detecte, por exemplo, inicializando uma variável compartilhada ou enviando uma mensagem. Entretanto esta comunicação somente ocorrerá se os eventos de “realizar a ação” e “deletar a ação” garantidamente ocorrerem nesta ordem. Assim sendo a sincronização entre tarefas pode ser vista como um conjunto de restrições que ordenam os eventos de modo a ocorrerem na sequência correta. O programador utilizada os mecanismos de sincronização para retardar a execução de uma tarefa de modo a garantir estas restrições /AND 83b/. A literatura de sistemas operacionais é farta em exemplos da necessidade de sincronização /HOLT 83//HOR 78//WIL 84/.

Na presente implementação por tratar-se de um sistema distribuído onde a comunicação e sincronização devem ser uniformes e independentes da distribuição das tarefas no sistema, foi adotado o mecanismo de mensagens passantes ou troca de mensagens.

As diretivas que suportam a troca de mensagens são:

a) SEND: para envio não bloqueante;
b) CALL: para envio bloqueante;
c) REC: para recepção bloqueante.

Por bloqueante entende-se a operação que suspende a execução da tarefa corrente colocando-a na fila de espera apropriada. No sistema aqui descrito tem-se duas filas associadas as diretivas de comunicação denominadas CALL_QUEUE e REC_QUEUE. Como o próprio nome indica cada uma das filas é utilizada por uma das diretivas. Pelo fato da diretiva SEND nunca bloquear a tarefa corrente, ela não necessita de fila de espera. Uma tarefa só é retirada da fila de espera se a operação solicitada é completada ou se o tempo de espera especificado é ultrapassado.

Além destas duas filas tem-se também uma área de memória destinada a alocação de buffers para armazenamento das mensagens. Deve ser ressaltado que as mensagens serão sempre armazenadas junto ao receptor ou seja, em buffers do núcleo onde reside a tarefa destino. No caso de uma interação remota (comunicação entre tarefas de estações diferentes) serão necessários buffers para armazenamento temporário das mensagens a serem transmitidas ou recebidas pelo subsistema de comunicação. Para facilitar a gerência, os buffers são encadeados em grupos conforme suas funções. Estes grupos são:

a) RECBUF: armazena mensagens a espera que a tarefa destino execute a diretiva REC. Por sua vez, sempre que executada, esta diretiva verifica neste grupo se a tarefa origem já envio a mensagem.
b) TXBUF: contém as mensagens a ser enviadas aos núcleos remotos.
c) RXBUF: onde são colocadas as mensagens recebidas pelo subsistema de comunicação. Estas mensagens tanto podem ser destinadas ao RECBUF, como serem mensagens de controle para o sistema.

Os buffers temporários são manipulados por duas tarefas do sistema que são a tarefa TXREM, destinada a transmissão das mensagens armazenadas em TXBUF para a estação de destino, utilizando para tanto o subsistema de comunicação através de seu protocolo de transporte, e a tarefa RXREM que verifica a existência de mensagens em RXBUF e se for o caso as transfere para o RECBUF.

A seguir são distribuídos os procedimentos realizados para execução das diretivas de comunicação e sincronização. Para maior clareza estes procedimentos são divididos em interações locais e interações remotas.

6.2.1 Interação local

Uma interação é dita local quando envolve tarefas residentes em uma mesma estação. Neste caso a comunicação se processa da seguinte maneira.

Quando é executado um CALL é verificado na REC_QUEUE se a tarefa destino já está esperando. Em caso positivo a mensagem é copiada para a tarefa destino que é retirada da fila de espera e colocada em estado apto (READY). Por sua vez a tarefa transmissora também é colocada na fila de aptas (READY). O detalhe aqui é que como a tarefa receptora é colocada em primeiro lugar, se sua prioridade for superior ou igual a tarefa transmissora ela será executada em primeiro lugar. Em caso negativo (isto é, se a tarefa destino não está esperando) mensagem é colocada em RECBUF e a tarefa corrente CALL_QUEUE sendo escalonada outra para seguir executando.

No caso do SEND, o procedimento é idêntico ao descrito acima com a diferença de que a tarefa origem nunca é bloqueada sendo entretanto colocada na READY após a execução da diretiva.

Na execução do REC, é verificado se a mensagem está no RECBUF. Se não está, a tarefa corrente é colocada na REC_QUEUE e escalonada outra para seguir executando. No caso da mensagem estar no RECBUF ela é copiada para a tarefa corrente. A seguir a tarefa origem é retirada da CALL_QUEUE e ambas, a tarefa corrente e a origem (nesta sequência) são colocadas na READY.

6.2.2 Interação remota

Uma interação remota envolve tarefas residentes em estação diferentes. Neste caso além dos procedimentos comuns a uma interação local são necessários outros de modo a permitir que a mensagem chegue ao núcleo destino e seja armazenada no RECBUF deste.

Na execução do CALL remoto a mensagem é colocada no buffer temporário TXBUF e a tarefa corrente na CALL_QUEUE, sendo escalonada outra. Para o SEND o procedimento é similar com a diferença de que a tarefa corrente não é bloqueada e sim colocada na READY.

Para o REC os procedimentos são os mesmos do caso da interação local, com a diferença de ser colocada uma mensagem de controle no TXBUF para o núcleo onde reside a tarefa origem, ou transmissoras, no caso de haver mensagens esperando.

Complementando estes procedimentos atuam duas tarefas do sistema que são TXREM e RXREM. A TXREM tem como única função verificar a existência de mensagens no TXBUF e, em caso positivo, enviá-las para os destinos. Por sua vez a tarefa e RXREM verifica a existência de mensagens no RXBUF. No caso da existência de uma mensagem esta é analisada para verificar se é destinada ao RECBUF ou se é uma mensagem de controle enviada por um REC remoto indicando que a tarefa de origem deve ser retirada da CALL_QUEUE e colocada na READY.

Para realização de suas funções estas tarefas se utilizam de subsistemas de comunicação. Com relação a este último cabe aqui ressaltar que todas as mensagens trocadas entre estações partem do TXBUF da origem e chegam ao RXBUF do destino.

6.2.3 As diretivas de comunicação

A seguir são apresentadas as diretivas de comunicação e sincronização.

- SEND (dest, est, msg, tam, tempo_esp): envio não bloqueante.
Dest: identificador da tarefa destino;
Est: indica a estação onde reside a tarefa destino;
Msg: é a mensagem, a qual pode ser qualquer sequência de bytes;
Tam: tamanho da mensagem “msg” em bytes;

Tempo_esp: indica o time-out. Em caso de falha devolve o código de erro. No caso especifico do SEND isto ocorrerá quando não haver buffer disponível.

- CALL (dest, est, msg, tam, tempo_esp): envio bloqueante.
- REC (orig, est, msg, tam, tempo_esp): recepção bloqueante.

Orig: identificador da tarefa de oriem. Caso seja “ANYONE” permite recepção de qualquer tarefa.

O parâmetro “est” é necessário por que no caso da estação não ser especificada pelo usuário é preciso que o núcleo mantenha um registro com o identificador de todas tarefas existentes no sistema. De qualquer forma ambas as alternativas levariam a uma maior complexidade do núcleo e aumentariam o tráfego de mensagens entre estações.

O tamanho da mensagem, em bytes, é indicado pelo parâmetro “tam”. Isto é necessário por ter sido decidido não impor nenhuma restrição quanto ao tamanho da mensagem. Desta forma o usuário poderá definir o parâmetro “msg” como sendo de qualquer tipo de dado permitido pela linguagem utilizada. No caso de interações remotas em que as tarefas tenham sido escritas em linguagens diferentes, deverão ser usadas estruturas de dados semelhantes. Isto é possível desde que os parâmetros sejam passados “by reference”. Desta forma basta que o núcleo copie os “tam” bytes de “msg” para a área de memória desejada, que poderá ser a área para armazenamento de mensagens ou área de dados da tarefa receptora.

Na prática poderá haver uma limitação quanto ao tamanho máximo permitido para a mensagem em função da política adotada para gerência das áreas para armazenamento das mensagens (RECBUF, TXBUF e RXBUF). Na implementação efetuada este tamanho é de 128 bytes sendo determinado pela constante TAMSG do módulo NUCLO.PAS.

6.2.4 O tratamento de erros

O tratamento de erros é realizado a nível de aplicativo sendo utilizado para tanto o parâmetro “TEMPO_ESP”. Este parâmetro possui dupla função. Na entrada

serve para indicar para o núcleo o tempo máximo que a tarefa corrente pode permanecer na espera do término da operação solicitada. Já na saída é por este parâmetro que é devolvido, se houver, o código de erro.

Este tipo de abordagem foi adotado pois simplifica as funções suportadas pelo núcleo, além de permitir ao usuário uma maior flexibilidade no tratamento de situações de erro, as quais podem ser atendidas por rotinas (procedures) específicas de cada tarefa ou então por uma rotina genérica a qual pode ser compartilhada pelas diversas tarefas do usuário em uma determinada estação. Esta última opção depende entretanto das características do compilador utilizado /ITH 81/.

Um exemplo típico de utilização de uma diretiva de comunicação com a previsão de uma situação de erro é apresentada a seguir:

```
CALL (..., tempo_esp);
if temp_esp (O then ERRO);
```

6.3 Controle de tempo

Estas diretivas permitem ao usuário o acesso ao relógio de tempo real do sistema de modo a possibilitar operações dependentes de tempo. São fornecidos recursos para manipulação da data e horário (inicialização ou leitura) além da possibilidade de uma tarefa atrasar sua execução por um certo período de tempo.

As diretivas para controle de tempo são:

- SETIME (hora, min., seg., data): inicializa hora e data.
- TIME (hora, min., seg., data): devolve hora e data.
- DELAY (qt_mil): permite atrasar a execução da tarefa por um certo período de tempo.

qt_mil: período de tempo milisegundos.

6.4. Manipulação de interrupções e operações de E/S

Por se tratar de um sistema de tempo real orientado para aplicações de controle de processos torna-se imprescindível o suporte a eventos ou interrupções de hardware.

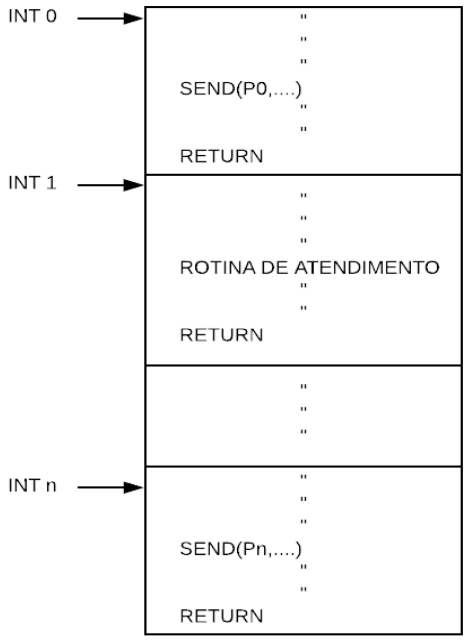
Para manipulação de eventos o núcleo possui condições de mapear cada interrupção (interrupção vetorada) para uma diretiva de comunicação, que envia uma mensagem para a tarefa de atendimento da mesma. Desta forma, o tratamento de interrupções pode ser realizado por tarefa do sistema escritas em alto nível. Esta facilidade entretanto não exclui a possibilidade de atendimento por rotinas do Kernel.

A fig. 1 apresenta o aspecto da tabela de mapeamento por interrupções. Pode ser observado que as interrupções INT 0 e INT n serão atendidas por tarefa do sistema, enquanto a INT 1 será atendida por uma rotina do núcleo.

As tarefas de atendimento de interrupções poderão ter a seguinte forma:

```
                    "
                    "
                    "
          Begin
                    loop forever
                              REC(ANYONE,....);
                              "atendimento da interrupção"
                    end loop;
          End.
```

Tabela 1. Tabela de mapeamento por interrupções



Fonte: autor, 1987.

Numa comparação dos dois casos pode-se dizer que o atendimento por tarefas do sistema é mais flexível a alterações, além de garantir que a tarefa de maior prioridade será escalonada no menor intervalo possível.

Já com a utilização de rotinas do núcleo a vantagem consiste em realizar o atendimento do evento num intervalo de tempo menor que no caso anterior (não é necessário enviar mensagem, nem realizar a troca de contexto do processador e o despacho). Por outro lado apresentada como desvantagem a necessidade de um maior cuidado na a elaboração das rotinas, visto que as mesmas integrarão o núcleo. Neste caso, obviamente, o núcleo será maior e consumirá maior tempo de execução.

O atendimento de operações de E/S é realizado utilizando as primitivas de sincronização e comunicação (CALL e REC) e uma tarefa específica para cada dispositivo a ser compartilhado pelas operações de E/S.

Estas tarefas são denominadas secretárias /DIJ 72/ e tem a função específica de sincronizar o acesso a um recurso compartilhado, garantindo a exclusão mútua no acesso ao mesmo. Sua função é semelhante a de um monitor /HOA 74/ podendo ser considerada como um monitor ativo.

7. DETALHAMENTO DA IMPLEMENTAÇÃO

7.1 Introdução

Nos capítulos anteriores foram descritas as características de um sistema distribuído de tempo real e foi apresentado um modelo para estruturação de tais sistemas, tendo sido proposto um sistema com as características descritas, implementável através de uma linguagem sequencial convencional cujo compilador permita a inclusão de rotinas externas. Finalmente foi descrito o núcleo do sistema o qual é o componente do software encarregado de suportar as características a serem acrescentadas ao ambiente original.

Neste capítulo é descrita uma implementação seguindo os conceitos e modelos acima mencionados.

7.2 A escolha do ambiente básico

Por ambiente básico entende-se aquele ambiente computacional ao qual serão acrescentados o núcleo e, eventualmente, o subsistema de comunicação.

Dentre as características básicas exigidas estava a de ser um microcomputador que possuísse uma linguagem de alto nível cujo compilador permitisse o uso de rotinas externas.

As opções existentes na época do início dos trabalhos (segundo semestre de 1984) eram o EDISA ED218 possuía compiladores Pascal e C. Além disto esta máquina utilizava o microprocessador Zilog Z80 o qual é bem mais popular em aplicação de controle de processos que o Motorola 6501 da linha Apple, além de ser mais poderoso. Outro fato a ser considerado é que o ED 281 utiliza o sistema operacional CP/M que pode ser tido como o padrão para micros de 8 bits, relativamente simples e bem documentado. Com isto fica facilitada a instalação do

sistema obtido em outros hardwares que utilizem o sistema operacional CP/M e processador Z80 ou similares.

Por estas razões decidiu-se usar o microcomputador ED281.

Quanto as linguagens, as opções da época eram Pascal e C. A opção pelo Pascal deveu-se ao fato da mesma ser mais adequada a aplicação em questões, conforme foi justificado no capítulo 5, além de ser uma linguagem melhor conhecida o que facilitaria a implantação. Além disto o compilador disponível denominado PASCAL/Z (anexo 2) da Ithaca Intersystem Inc. apresentava as características necessárias entre as quais destacam-se:

- Permite a inclusão de rotinas externas;
- Compilação em separado;
- Extensões para aplicações científicas (ponto flutuante);
- Geração de código macro assembler para o Z80 (O que facilitou a multiprogramação do sistema);
- Geração de código reentrável;
- Eficiência com relação ao código gerado.

Assim, o sistema básico ficou sendo o microcomputador ED281 sistema operacional CP/M e compilador PASCAL/Z.

7.3 A estrutura do sistema

No anexo 1, encontra-se uma descrição dos sistema operacional CP/M bem como os motivos de não adoção do MP/M. No anexo 2 tem-se uma descrição compilador PASCAL/Z.

Conforme já foi mencionado, o modelo para implementação é o apresentado no capítulo 4. Desta forma, partindo do sistema básico ou original que consiste do nível aplicativo e do nível de sistema operacional (CP/M), núcleo e subsistema de comunicação.

No presente trabalho o subsistema de comunicação não foi implementado. Isto foi decidido pelo fato de na época não se dispor de equipamentos apropriado

(micros interligados) e pelo fato do trabalho necessário ficar um tanto exagerado para uma dissertação de mestrado.

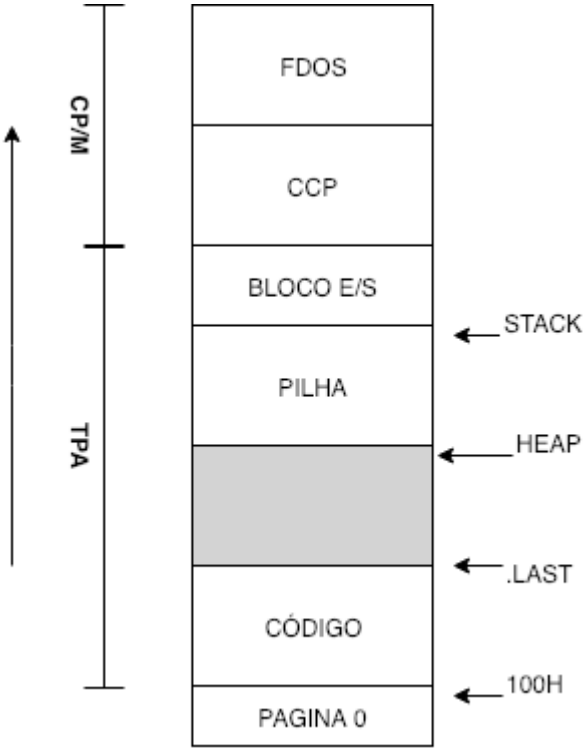
Desta forma o sistema fica reduzido aos níveis de aplicativo, sistema operacional e núcleo.

7.4 A multiprogramação do sistema

A principal limitação do ambiente básico disponível era o fato do mesmo não ser um ambiente multitarefas. A possibilidade do uso do MP/M foi descartada devido a sua segmentação fixa de memória determinada durante a geração do sistema (apêndice 1), o que torna o sistema lento devido aos acessos ao disco (para o Swap), além de apresentar problemas de fragmentação de memória. Outro fato a ser mencionado é que a adoção do MP/M (na realidade um sistema operacional tipo tempo compartilhado) acarretaria uma disciplina mais rígida com relação as características do sistema a ser desenvolvido.

Desta forma a viabilização da implementação dependeria da possibilidade da multiprogramação do código gerado pelo compilador escolhido. Neste ponto o compilador Pascal/Z revelou-se adequado pelas suas características de gerar macro código assembly Z80 reentrável. O fato de ser reentrável garantia a separação do código de cada programa (tarefa) em dois componentes: um segmento de código puro e um segmento de dados (pilhas). Assim sendo para multiprogramação seria necessário dividir o espaço de endereçamento disponível no CP/M para os programas do usuário (TPA) em dois segmentos: um deles contendo todos os segmentos de código das tarefas, e outros contendo as pilhas de cada tarefa.

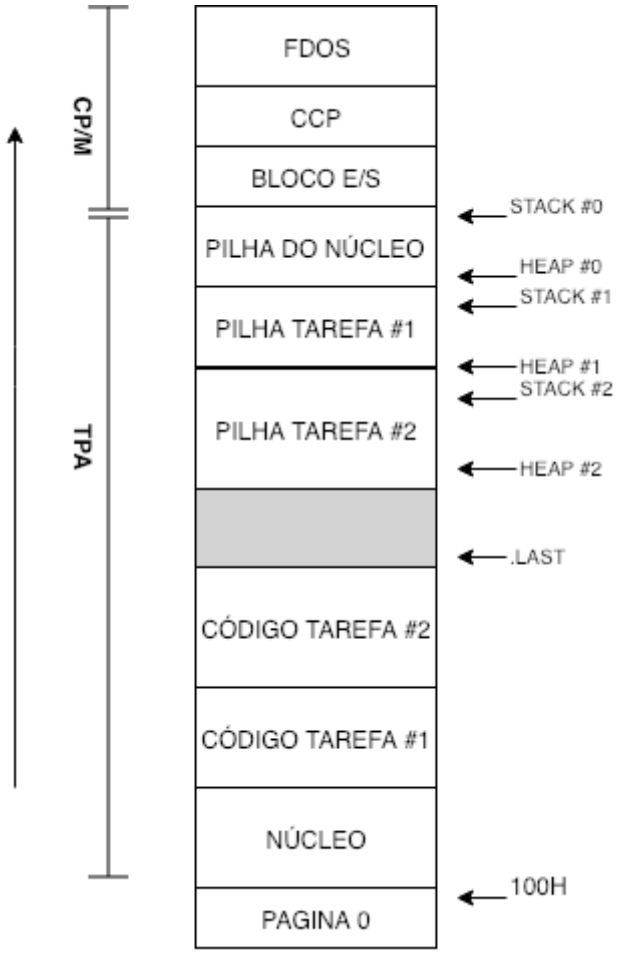
Figura 5. Imagem da memória de um programa Pascal/Z



Fonte: autor, 1987

A figura 5. Apresenta a imagem da memória de um programa Pascal/Z rodando em um ambiente CP/M. Como pode ser visto, tem-se três componentes: o bloco de E/S, utilizado para operações com o disco; a pilha, delimitada por dois ponteiros, STACK (início) e HEAP (fim); e o segmento de código.

Figura 6. Imagem da memória do Pascal/Z multiprogramado para o caso de duas tarefas



Fonte: autor, 1987.

Com a multiprogramação a imagem de memória para o sistema fica como apresentado na figura 6. Assim tem-se: o bloco de E/S (cada tarefa possui o seu bloco); as pilhas de dados, delimitadas por um par de apontadores; e por fim os segmentos de código agrupados na região inicial da memória. O final da área de código é indicado pelo ponteiro .LAST gerado pelo ligador. O núcleo apesar de não ser uma tarefa (não concorre pelo processador) possui sua própria pilha e seu segmento de código obrigatoriamente deve iniciar o segmento de código do sistema.

O bloco de E/S de cada tarefa é salvo junto ao descritor da mesma e carregado no topo da TPA toda vez que a tarefa é despachada. Isto é necessário

pois esta área deve se situar no topo da TPA pois é acessada por rotinas intrínsecas do PASCAL/Z. Sua função é atuar como um buffer para operações com o disco.

7.4.1 Características do macro-código

Todo código (tarefa) gerado pelo PASCAL/Z começa pela inicialização de sua pilha. Isto é feito pela macro denominada ENTR que faz parte da biblioteca de macros do Pascal/Z (MAIN.SRC), usada durante a montagem do código gerado pelo compilador (apêndice 2).

Desta forma para a multiprogramação é necessário alterar esta macro de modo que cada tarefa inicialize sua pilha de maneira sequencial na área de memória destinada para tanto.

Como pode ser visto o código para inicialização da pilha é parte da tarefa e não do núcleo. Desta forma para inicializar uma tarefa a diretiva do núcleo (START) desvia para a rotina .ENTR da tarefa sendo inicializada, a qual após inicializar a sua pilha retorna para o núcleo.

7.5 Módulos componentes do sistema

O sistema é composto pelo núcleo e seus diversos componentes e pelas bibliotecas de macros usados em tempo de montagem. Estas bibliotecas contém os códigos das macros cujas chamadas encontram-se no código gerado pelo compilador Pascal/Z.

7.5.1 As bibliotecas de macros

No total tem-se três bibliotecas de macros assim denominados:

a) NUCBLI.SRC: é a biblioteca usada para montagem do núcleo;
b) MAIN. SRC: usada para montagem das tarefas;
c) EMAIN. SRC: usada para montagem dos módulos das tarefas quando for usada compilação em separado.

Estas bibliotecas foram desenvolvidas baseadas nas que acompanham o compilador Pascal/Z.

A biblioteca EMAIN.SRC é idêntica a original. A MAIN. SRC apesar do nome ser idêntico ao existente no pacote do compilador, na realidade é baseada na EMAIN.SRC original com alterações na macro .ENTR para realizar a multiprogramação. O nome foi mantido para fins de documentação pois ela continua sendo usada do mesmo modo como consta na documentação do compilador. Finalmente tem-se a biblioteca NUCBLI.SRC que é usada para montagem do núcleo do sistema. Esta biblioteca é baseada na MAIN.SRC original com alterações que permitem a inicialização das estruturas de dados do núcleo e preparação do ambiente para iniciar a multiprogramação.

7.5.2 Os componentes do núcleo

O software que constitui o núcleo é dividido em três componentes distintos que são:

a) NÚCLEO.PAS: é o núcleo do sistema;
b) DIR.PAS: contém as diretivas escritas em Pascal;
c) DIRASM.SRC: contém as diretivas escritas em Assembly e outras rotinas para suporte dos módulos em Pascal.

A razão para divisão da parte escrita em Pascal em dois módulos é para permitir a geração automática pelo compilador dos pontos de entrada (entry points) para as diretivas.

8. DETALHAMENTO DO NÚCLEO

Este capítulo apresenta os principais componentes do núcleo. São apresentadas as estruturas de dados utilizadas e as diretivas (ou primitivas) invocáveis pelas tarefas.

Descrições mais detalhadas destes componentes encontram-se nas listagens dos mesmos no apêndice 1 deste trabalho. No apêndice 3 estão os fluxogramas das diretivas de comunicação.

8.1 Estruturas de dados do núcleo

O núcleo deve manter um conjunto de estruturas de dados de forma a suportar o ambiente de execução desejado.

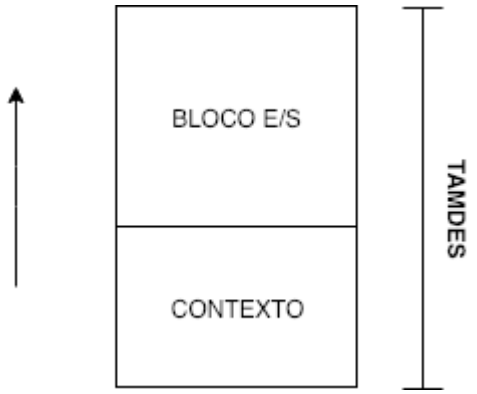
A seguir são apresentadas as estruturas de dados principais, usadas pelos diversos módulos componentes NUCLEO.PAS, DIRPAS.PAS e DIRASM.SRC com destaque para as utilizadas por este último. Estas estruturas são: o descritores de tarefas, área para alocação de descritores, tabelas e tarefas inicializadas, filas de tarefas e área para armazenamento de mensagens.

8.1.2 O descritor de tarefas

O descritor de uma tarefa tem por objetivo armazenar todas as informações necessárias à restauração do contexto da tarefa quando esta perder o processador.

O descritor é dividido em duas partes que são: o contexto e o bloco E/S (figura 7). TAMDES indica o tamanho de descritor em bytes.

Figura 7. O descritor de tarefas

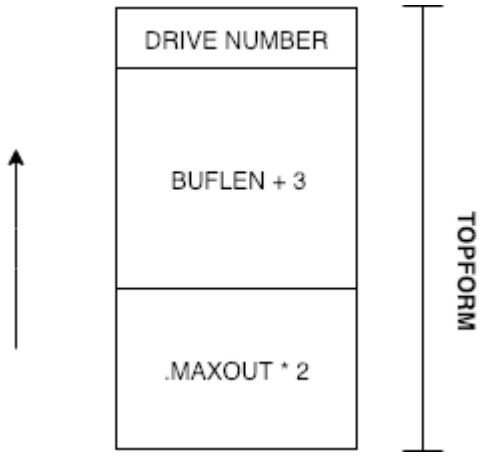


Fonte: autor, 1987.

O bloco E/S, figura 8, tem como função armazenar o número da unidade de disco corrente, informações referentes aos arquivos abertos e uma área (buffer) para operações com o disco. O bloco E/S é carregado no topo da TPA quando a tarefa recebe o processador. O tamanho do bloco de E/S é indicado por TOPFORM sendo:

TOPFORM = BUFLEN + 3+ (.MAXOUT * 2)
Onde:
BUFLEN: tamanho do buffer de E/S;
.MAXOUT: número de arquivos de saída.

Figura 8. O bloco E/S



Fonte: autor, 1987.

A área correspondente ao contexto, figura 9, armazena todos os registradores do processador (Z80) com exceção do contador de programa (PC). Além disto tem ainda as seguintes entradas:

NEXT: ponteiro para o próximo da lista;

LAST: ponteiro para o anterior da lista;

PRIORIDADE: a prioridade da tarefa;

MIX: número de identificação interna.

Figura 09. O contexto

MIX
PRIORIDADE
S
P
H'
L'
D'
E'
B'
C'
A'
F'
H
L
D
E
B
C
A
F
I
Y
I
X
LAST (ANTERIOR)
NEXT (PRÓXIMO)

Fonte: autor, 1987.

Os ponteiros são usados para encadear o descritor na fila de tarefas aptas ou nas outras filas existentes. O MIX corresponde a um número no intervalo de 0 a NUMTAR-1 onde NUMTAR é o número máximo de tarefas do sistema. Este número é distribuído a tarefa durante sua inicialização.

O contador do programa (PC) não é salvo no descritor da tarefa mas sim no topo da pilha da mesma.

8.1.3 Área para alocação de descritores

Os descritores das tarefas são alocados em uma área contigua de memória, figura 10, sendo o tamanho da área (TAMARD) determinado pelo número máximo de tarefas permitidas, especificadas em NUMTAR.

Desta forma tem-se que:

TAMARD = TAMDES * NUMTAR

Onde:

TAMARD: tamanho da área para alocação de descritores;

TAMDES: tamanho do descritor.

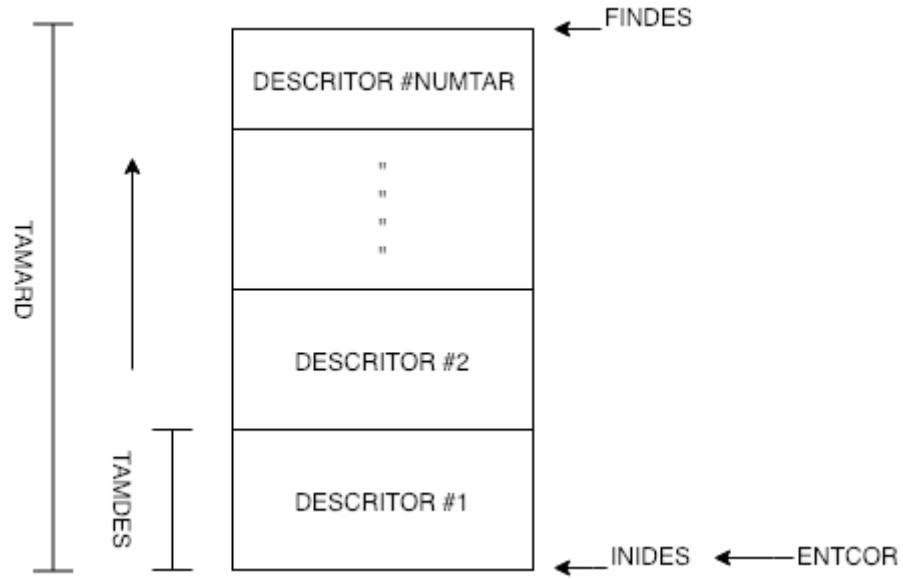
Tem-se ainda:

INIDES: início da área;

FIMDES: final da área;

ENTCOR: entrada corrente. Aponta para o próximo descritor a ser alocado.

FIGURA 10. A área para alocação de descritores



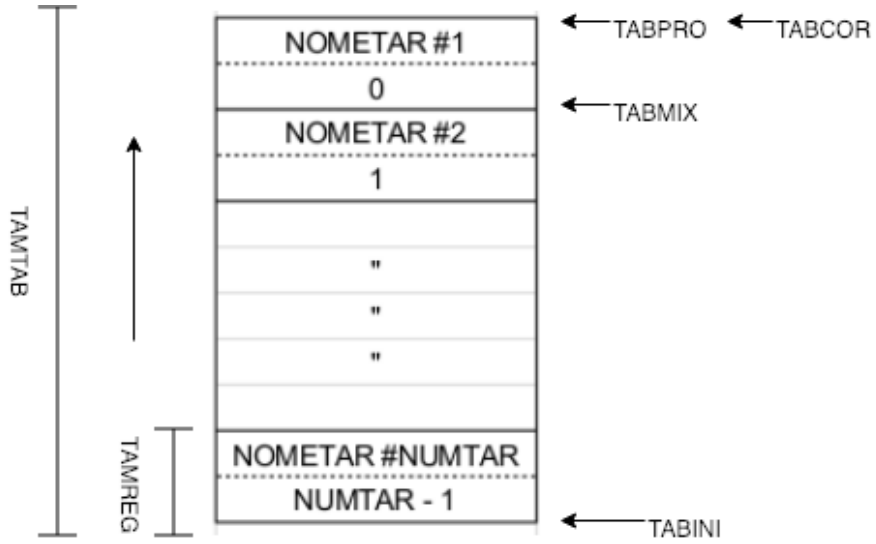
Fonte: autor, 1987.

8.1.4 A tabela de tarefas inicializadas

Nesta tabela constam os identificadores de todas as tarefas inicializadas pelo sistema e os seus números de identificação interna (MIX). A ordem de entrada na tabela é que determina o MIX associado à tarefa. A tabela 2 apresenta o aspecto da tabela onde tem-se:

NOMETAR: identificador da tarefa (máximo 6 caracteres);
MIX: número de identificação interno;
TABPRO: início lógico da tabela (final físico);
TABINT: final lógico da tabela (início físico);
TABCOR: entrada corrente;
TABMIX: entrada auxiliar para inicialização do MIX;
TAMREG: tamanho de um registro da tabela de bytes;
TAMTAB: tamanho da tabela (bytes).

Tabela 2. A tabela de tarefas inicializadas



Fonte: autor, 1987.

8.1.5 As filas de tarefas

O núcleo mantém quatro filas de tarefas: uma fila de tarefas aptas (ready), uma para tarefas bloqueadas ao executarem a diretiva REC (REC_QUEEU), uma para as tarefas bloqueadas ao executarem a diretiva CALL (CALL _QUEEU), e uma associada ao relógio, usada tanto para as tarefas que executam a diretiva DELAY (DELAY_QUEEU) como para implementar o “Time-out” das diretivas de comunicação.

A fila de tarefas aptas encadeia as tarefas que estão aptas para receber o processador. Esta fila é organizada pela prioridade das tarefas e pela ordem de entrada da mesma. Esta fila tem um “ponteiro” que indica a tarefa de mais alta prioridade e de entrada mais antiga na fila. Existe também na fila uma tarefa fictícia (IDDLE) que possui prioridade mais baixa que qualquer outra tarefa e que nunca se bloqueia. Isto facilita o trabalho do escalonador que sempre terá pelo menos uma tarefa para selecionar (a fila de aptas nunca ficará vazia).

Esta fila de tarefas aptas é na realidade uma lista duplamente encadeada cujos nodos são os descritores das tarefas. Para o encadeamento são usadas as entradas NEXT e LAST do descritor.

É interessante ressaltar que a ordenação das tarefas é feita durante a inserção na fila. Desta forma a seleção da tarefa a ser despachada se resume em pegar a primeira tarefa da fila.

As filas de tarefas bloqueadas não possuem duplo encadeamento pois as inserções sempre são feitas no final. A retirada das tarefas da fila não segue uma ordem (FIFO, por exemplo) pois as tarefas estão condicionadas a eventos como, modificação de variáveis globais e invocação de uma diretiva.

A fila de espera (DEL_QUEEU) possui uma ordenação determinada pela passagem do tempo. Entretanto no caso de uma tarefa ser desbloqueada pela efetivação de uma comunicação entre tarefas, esta será retirada da fila. No caso da tarefa bloqueada a espera de comunicação ser esta será retirada da fila de espera pelo esgotamento do tempo (“time-out”) o código de erro correspondente será marcado.

8.1.6 Área para armazenamento de mensagens

A área para armazenamento das mensagens é alocada dinamicamente pelo núcleo no caso em que a tarefa receptora não se encontre bloqueada a espera de mensagem. Neste caso a mensagem é copiada diretamente para a área de dados da tarefa receptora. Para simplificação da política de alocação e gerência da área para armazenamento de mensagens esta é constituída por buffers de tamanho fixo. O tamanho do buffer determina o tamanho máximo permitido para as mensagens. Este valor é determinado pela constante TAMSG do módulo NUCLEO.PAS. Os buffers disponíveis para armazenamento são encadeados em uma lista de buffers livres. Quando alocado, um buffer é inserido na RECBUF que é a lista de mensagens a espera da invocação da diretiva REC pela tarefa receptora.

8.2 As diretivas do núcleo

Nesta seção são apresentadas as diretivas do núcleo que podem ser acessadas ou invocadas de fora do mesmo. O formato de apresentação é o seguinte: resumo da finalidade, forma de invocação (chamada), parâmetros são passados através da pilha de tarefa invocante. No caso das diretivas escritas em Pascal (todas com exceção de START e QUIT) é necessário realizar alguns ajustes. Tais ajustes são realizados pelos chamados “pseudo-diretivas” e tem por finalidade copiar os parâmetros da pilha do núcleo, desempilhar os parâmetros da pilha da tarefa invocante, colocar o endereço de retorno (contador de programa) no topo da pilha da tarefa invocante e atualizar o ponteiro (registrador IY do Z80) para a área de dados do núcleo. Uma vez feitos estes acertos é chamada a diretiva propriamente dita.

8.2.1 Diretiva START

É a diretiva que realiza a inicialização das tarefas; ela inicializa as pilhas e as estruturas de dados do núcleo associadas às tarefas.

- Invocação:
START (proc.id, priorid, endepro);
- Parâmetros:
proc.id: identificador de tarefas;
priorid: a prioridade da mesma;
endepro: endereço inicial (absoluto) da área de código.

Procedimentos:

1. Aloca descritor para a tarefa;
2. Aloca entrada na tabela de tarefa inicializadas;
3. Inicializa pilha e descritor da tarefa;
4. Insere tarefa na fila de aptas;
5. Retorna controle para o procedimento inicializador.

8.2.2 Diretiva QUIT

Encerra a execução de uma tarefa. Esta diretiva é invocada de modo implícito pois sua chamada foi colocada na macro FINE (MAIN.SRC) que sempre encerra o código gerado pelo compilador. Desta forma, apesar de ser possível chama-la, esta primitiva não necessita ser invocada explicitamente a nível de tarefa de usuário.

Invocação:
QUIT:
Parâmetros:
“Nenhum”.

Procedimentos:
1. Anula entrada na tabela de tarefas inicializadas;
2. Realiza o despacho da tarefa de maior prioridade;
3. Passa o controle para a tarefa despachada.

8.2.3 Diretiva SEND

Diretiva de comunicação responsável pelo envio não bloqueante de mensagem da tarefa origem para a tarefa destino. Neste tipo de comunicação a tarefa origem, ou transmissora, nunca é bloqueante. A mensagem é armazenada na área destinada para tanto e a tarefa volta a fila de aptos.

Invocação
SEND (dest, est, msg, tam, tempo_esp);

Parâmetros:
Dest: identificador da tarefa destino;

Est: estação onde reside a tarefa destino;

Msg: mensagem;

Tam: tamanho de “msg” em bytes;

Tempo_esp: serve para devolver um código em caso de ocorrência de erro.

Procedimentos:

1. Se destino é local;

Então

1.1.1 Obtém o MIX do destino;

1.1.2 Se o destino estiver esperando pela mensagem;

Então

- copia mensagem para o destino;
- coloca origem na fila de aptos;
- retira destino da fila de bloqueadas;
- coloca destino na fila de aptas;

Senão

- aloca área para armazenar mensagem;
- coloca mensagem na área alocada;
- coloca origem na fila de aptas;

Senão

1.2.1 coloca mensagem na área para mensagens externas (TXBUF);

1.2.2 Coloca origem na fila de aptas;

2. Fim.

8.2.4 Diretiva CALL

É a diretiva responsável pelo envio bloqueante de mensagens permitindo a sincronização entre tarefas. Neste tipo de comunicação a tarefa origem é bloqueada até que a tarefa destino invoque a diretiva de recepção (REC).

Invocação

CALL (dest, est, msg, tam, tempo_esp):

Parâmetros:

dest, est, msg, tam: análogos a diretiva SEND;

tempo_esp: especifica o tempo de espera máximo no qual a tarefa pode ficar esperando pela efetivação da comunicação. No caso da ocorrência de erro serve para devolução do código do mesmo.

Procedimentos:

1. Se destino é local;

Então

1.1.1 obtém o MIX do destino;

1.1.2 se destino existe

Então

1.1.2.1 se destino estiver esperando pela mensagem;

Então:

- copia mensagem para o destino;
- retira destino da fila de bloqueadas;
- retira destino da fila de espera;
- coloca destino na fila de aptos;

Senão

- aloca área para armazenar mensagem;
- coloca mensagem na área alocada;

1.1.3 coloca origem na fila de bloqueadas;

Senão

- coloca mensagem na área para mensagem externas (TXBUF);
- coloca origem na fila de bloqueadas;

2. Fim.

8.2.5 Diretiva REC

É a diretiva para recepção das mensagens enviados pelas diretivas SEND ou CALL. A tarefa invocante sempre é bloqueada até que a comunicação seja efetivada.

Invocação:
REC (orig, est, msg, tam, tempo_esp)

Parâmetros:
Orig: identificador de tarefa origem. Caso seja “ANYONE” permite recepção de mensagens enviadas por qualquer tarefa.

est, msg, tam, tempo_esp: análogos aos parâmetros da diretiva CALL.

Procedimentos:

1. Obter seu MIX; /* seja t a identificação da tarefa */
2. Se não tiver mensagem t

Então

- Insere t na fila de bloqueadas;
- Insere t na fila de espera;

Senão

- Coloca t na fila de aptos;
- Copia mensagem da área de armazenamento (RECBUF);
- Se foi envio bloqueante

Então

Se foi externa

Então

- Coloca resposta na área para mensagem externas (TXBUF);

Senão

- Retira origem da fila de bloqueadas;
- Retira origem da fila de espera;
- Coloca origem na fila de aptas;

- Libera a área de armazenamento da mensagem

3. Fim.

8.2.6 Diretiva DELAY

Permite que a tarefa invocante atrase sua execução pelo período de tempo passado como parâmetro.

Invocação

DELAY (qt_mil);

Parâmetros:

qt_mil: período de tempo em milissegundos;

Procedimentos:

1. Coloca tarefa invocante na fila de espera;
2. Fim.

8.2.7 Diretiva SETIME

Inicializa a hora e data do sistema.

Invocação:

SETIME (hora, min, seg, data)

Parâmetros:

Hora: hora a ser Inicializada;

Min: minutos;

Data: data, no formato DDMMAA.

Procedimentos:

1. Coloca tarefa invocante na fila de aptas.
2. Inicialização globais do objeto.
3. Fim.

8.2.7 Diretiva SETIME

Invocação:

TIME (hora, min, seg, data)

Parâmetros:

Análogos aos parâmetros de SETIME.

Procedimentos:

1. Coloca tarefa invocante na fila de aptas.
2. Obtém endereço absoluto dos parâmetros passados.
3. Copia hora e data correntes para o endereço absoluto obtido.
4. Fim.

9. CONCLUSÃO

O trabalho cumpriu com o seu objetivo de conceituar e implementar um ambiente voltado à construção de sistemas de controle distribuído.

O sistema implementado apresentada a grande vantagem de permitir a criação de ambientes computacionais distribuídos utilizando meios normalmente disponíveis em ambientes convencionais. Desta forma todos os utilitários existentes no ambiente original (editores, compiladores, ligadores, montagens, etc...) podem ser usados na construção dos aplicativos. Isto representa uma vantagem não só pela dispensa da construção de tais ferramentas, como pela facilidade de aprendizado visto que muitas vezes os usuários já dominam essas ferramentas.

Por ser um sistema multiprogramado, com a utilização de um processador de 8 bits e 64 Kbytes de endereçamento pode ocorrer que a área de memória destinada aos aplicativos fique pequena se comparada a área destinada ao sistema, restringindo o número e o tamanho das tarefas do usuário. Um melhor desempenho pode ser obtido pela implementação do sistema em micros de 16 bits como os da linha IBM PC, por exemplo.

O fato da implementação ter sido efetuada em uma só estação não a invalida pois o trabalho constitui-se em uma boa experiência na medida que os conceitos propostos puderam ser verificados quanto a validade e dificuldade de implementação.

Para obtenção de um sistema distribuído seria necessária a definição da estrutura do mesmo quanto ao número de estações, tipos das estações e do subsistema de comunicação. Uma opção interessante seria a implementação de uma versão em uma máquina de 16 bits e conexão à estação já existente por meio de protocolos simplificados. O modelo apresentado não obriga a que todas estações sejam idênticas. É necessário apenas que os protocolos do subsistemas de comunicação de cada estação sejam compatíveis e que a semântica das mensagens seja a mesma.

A implementação foi efetuada de maneira modular de forma que as novas características pudessem ser desenvolvidas, validadas e acrescentadas ao sistema separadamente. Por este fato e pela documentação disponível ficam facilitadas modificações e revisões futuras.

Apesar do sistema ser relativamente simples, espera-se que o mesmo seja bastante útil para suporte dos trabalhos a serem desenvolvidos, sendo uma alternativa interessante para o desenvolvimento e pesquisa de sistemas distribuídos, pelo menos até que se difundam linguagens específicas para este fim.

10. REFERÊNCIAS

/ALL 86/ ALLAN, Roger. Factory Comunication: MAP promisses to pul the pieces together.

Eletronic Design, New Jersey, 34 (10, 11,12): 103-112, may 1986.

/AND 81/ ANDREWS, G. R. Sinchronizing resources. ACM Trans. on Programing Languages and System, New York, 12 (8): 719-53, aug. 1982.

/AND 83/ ANDREWS, Gregory R. & Surveys. FRED, B. Concepts ad Notations for Concurrent Programing. Computing System, New York, 15 (1): 719-53: 3-43, mar. 1983.

/CAN 84/ CANTO, C. E. P. do & Walter, C. Estruturando o núcleo de sistemas distribuídos. In: SIMPÓSIO SOBRE DESENVOLVIMENTO DE SOFTWARE BÁSICO 4, São José dos Campos, out. 29-31, 1984. Anais. São Paulo, p. 173-183.

/CAN 84/ CANTO, C. E. P. do et alii. Uma proposta para construção de sistema distribuído. In: CONGRESSO NACIONAL DE AUTOMAÇÃO INDUSTRIAL 2, São Paulo, nov. 25-29, 1985. Anais. São Paulo, p. 295-298.

/CHE 84/ CHERITON, David, R. The V Kernel: A Software Base for Distributed System. IEEE Software, Los Alamitos, 42, april 1984.

/DIJ 72/ DIJKSTRA, E. W. Cooperating sequencial processes. In: F. Genys (ED.), Programming Languages. New York, Academic Press, 1972

/DOO 85/ DOOL, Gary. Local área Networks Can Improvise Efficiency of Finishing Operations. Pulp & Paper, july 1985, p. 78-80.

/ENS 78/ ENSLOW JR., Philip H. What is a “Distributed” Data Processing System? Computer, Long Beach, 11 (1): 13-21, jan. 1978.

/FEU 81/ FEUER, A. R. & GEHANI, N. H. A comparison of the programming languages C and Pascal. ACM Computing Surveys, New York, 14 (1): 73-92, mar. 1981.

/FOL 85 / FOLEY, Jenold S. The status na direction of open systems interconnection. Data Comunications, New York, 14 (2): 177-193, fev. 1985.

/HAN 73/ HANSEN, Per Brinch. Concurrent Programming Concepts. Computing Surveys, New York, 5 (4): p. ?, dez. 1981.

/HAN 78/ HANSEN, Per Brinch. Distributed Process: A Concurrent Programming Concepts. Comunications of ACM, New York, 21 (11) 934-941, nov. 1978.

/HEM 84/ HEMINGUAY, C. Sucsessful Pascal programs stress easy portability. EDN. Boston, v. n.:205-214, 9 fev. 1984.

/HOA 74/ HOARE, C. A. R. Monitors: An Operating System Structuring Concept. Communication of the ACM, New York, 17 (10): 549-557, oct. 1974.

/HOA 78/ HOARE, C. A. R. Communication Sequencial Processes. Communication of the ACM, New York, vol. 21 (8): 666-777, aug. 1978.

/HOD 84/ HODGSON, Ralph. Programming the 68000 in high-level language for VME. Microprocessors and Microsystem, Guildford, 8 (7): 338-349, set. 1984.

/HOG 81/ HOGAN, Thom. Osborne CP/M User Guide. Bekerley, OSBORNE/MacGraw-Hill, 1981.

/HOL 83/ IMOTO, R. C. Concurrent Euclid. The Unix System and Iunis. Addison-Wesley. Ontario,1983.

/HOL 83/ IMOTO, Jaime Itaru. Automação de plantas industriais e distribuição de inteligência: princípios e práticas. In: CONGRESSO NAICONAL DE AUTOMAÇÃO INDUSTRIAL 1, ANAIS. São Paulo, 1983, p. 25-33.

/INT 80/ INTERNACIONAL ORGANIZATION POR STANDARDIZATION. Data Processing = Open System Interconnection: Basic Reference Model. New York, 1980. (ISO/TC97 / SC16).

/ITH 80a/ ITHACA INTERSYSTEMS, INC. ASMBLE/Z. A Relocating Macro Assembler. New York, 1980.

/ITH 80b/ ITHACA INTERSYSTEMS, INC. A Linking Loader. New York, 1980.

/ITH 81/ ITHACA INTERSYSTEMS, INC. PASCAL/Z User's Manual. New York, 1981.

/JON 78/ JONES, Anita K. et alii. StarOS. A Multiprocessor Operating System for the Support of Task Forces. In: SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 7, Pacific Grove, Dec. 10-12, 1979. Proceedings. New York, ACM, 1979, p. 117-127.

/KRA 84/ KRAMER, J. et alii. The conic programming language version 2.4. Imperial College. London, 1984. (Research Report DOC 84/19).

/LEB 84/ LEBLANT, Thomas J. et alii. The StarMod Distributed Programming kernel. Software Practice and Experience. New York, 14 (12): 1123-1139. Dec. 1984.

/QUE 84/ QUEIROZ, J.A.M. DE, Um Modelo de sistema distribuído e proposta de kernel para REDURGS, Porto Alegre, CPGCC da Universidade Federal do Rio Grande do Sul – UFRGS, 1984 (Dissertação de metrado).

/SAV 85/ SAVITZKY, Stephen. Real=time Microprocessors Systems. New York, Van Nostrand Reinhold, 1985.

/SCH 84/ SHOEFFER, JAMES D. Distributed Computer System for Industrial Process Control. Computer, Long Beach, 17 (2): 11-18, febr. 1984.

/STA 84/ STALLINGS, William. Local Network. Computing Surveys. New York 16 (1): mar. 1984.

/WAT 80/ WATSON, Richard W. Distributed System Architecture Model. In: ADVANCE COURSE ON DISTRIBUTED SYSTEM – ARCHITECTURE AND IMPLEMENTATION, Munchen, mar. 4-13, 1980. Berlin, Spring-Verlag, 1981. Cap. 2, p. 10-43 (Lecture Notes in Computer Sciense, 105).

/WIL 84/ WILLIANSON., Ronald & HOROWITZ, Ellis. Concurrent Comunication and Synchronization Mechanisms. Software=Pratice and Experience. New York, 14 (2): 135-151, fev. 1984.

/XER 80/ XEROX CORPORATION, INTE CORP. AND DIGITAL EQUIPMENT CORP. The Ethernet: local área networks, data linha layer and physical layer specifications version 1.0. 30 sep. 1980.

(YOU 78) YOUNG, S. J. A Survey of real time programming languages. Control Systems Centre Report No. 424. University of Manchester, Manchester, aug. 1978.

APÊNDICES

APÊNDICE 1
O SISTEMA CP/M

1.1. Introdução

O CP/M (Control Program for Microcomputers) é um sistema operacional de disco produzido pela Digital Research tendo sido originalmente desenvolvido para uso em sistemas computacionais baseados nos microprocessadores Intel 8080 e posteriormente Zilog Z80. Pelas suas características onde se destacam confiabilidade, portabilidade, e facilidade de uso (user friendly) o CP/M tornou-se o operacional mais difundido para microcomputadores de 8 bits baseados nas tecnologias Intel e Zilog.

1.2. A família CP/M

A grande aceitação do CP/M propiciou o surgimento de uma verdadeira família de sistema onde tem-se o CP/M propriamente dito, o MP/M (multiusuário) e o CP/NET (rede de sistemas CP/M).

Neste trabalho será considerado somente o sistema CP/M e será feita uma exposição dos motivos que levaram a não utilização do sistema MP/M.

1.2.1 Portabilidade

O CP/M foi concebido de modo a ser dividido em duas partes visando facilitar sua instalação nos mais diversos microcomputadores: uma parte invariante e uma parte variante. A parte invariante contém o sistema operacional de disco escrito na linguagem PL/M. A parte variante, escrita na linguagem assembly da máquina onde

será instalada, contém as rotinas de E/S requeridas pelo hardware em questão. Esta divisão possibilitou a grande difusão do CP/M.

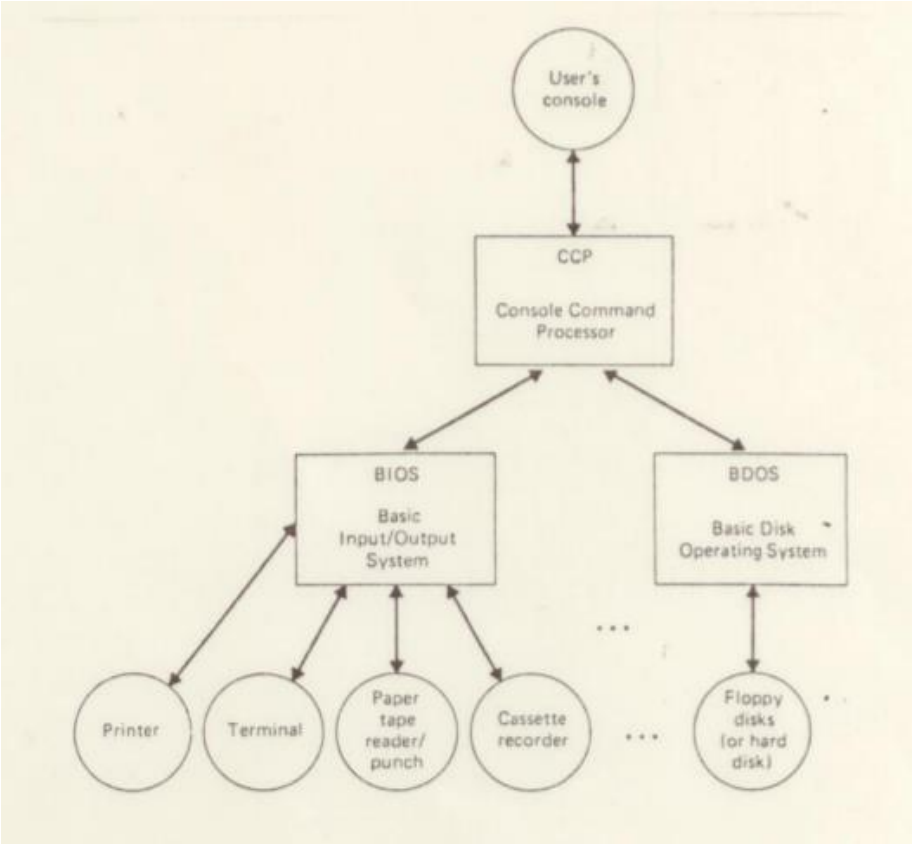
1.3. A estrutura do CP/M

O CP/M consiste em três subsistemas (figura 1.1.):

- O Console Comand Processador (CCP)
- O Basic Input/Output System (BIOS)
- O Basic Disk Operating System (BDOS)

O CCP interpreta os comandos do usuário e envia as repostas. É ele quem cria a visão que o usuário tem do microcomputador – uma visão “user friendly” que permite ao usuário não necessitar conhecer o funcionamento interno do CP/M e do computador. O CCP se utiliza do BIOS e do BDOS para realizar a entrada e saída e o processamento dos arquivos.

Figura 1. Os componentes do CP/M



Fonte: autor, 1987

Os bios contém as diversas rotinas que enviam e recebem dados dos dispositivos bem como informações a respeito do estado destes e do sucesso ou falha de uma operação de E/S. O BIOS consiste na chamada parte variante do CP/M.

O BDOS contém as diversas rotinas para gerência dos discos tornando esta transparente ao usuário.

1.3.1. Alocação de memória

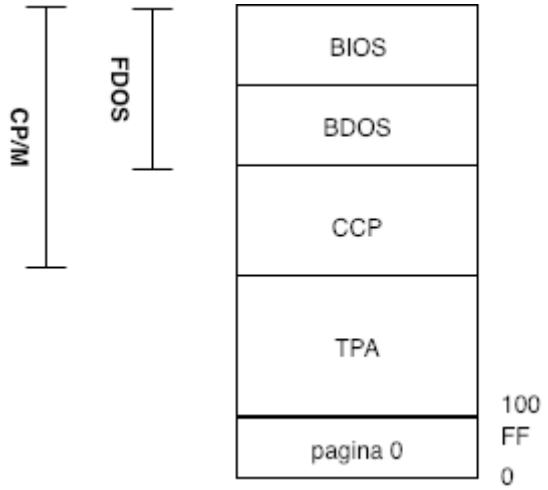
O CP/M é um sistema que opera com memória real. Certas áreas de memória são reservadas para o sistema, mas a grande maioria fica disponível ao usuário.

A figura 1.2 mostra a utilização de memória por um sistema CP/M típico. Os primeiros 256 bytes (página 0) são reservados para os diversos parâmetros do

sistema ficando o sistema operacional propriamente dito na parte superior do espaço do endereçamento. O espaço restante, denominado Transient Program Area (TPA), fica disponível para o usuário. Na terminologia CP/M é comum referenciar-se a combinação do BIOS e BDOS como o FDOS (Functional Disk Operation System). O FDOS e o CCP juntos constituem o CP/M.

No caso do usuário necessitar de mais espaço do que o disponível na TPA é possível avançar sobre a área do CP/M correspondente ao CCP restando somente o FDOS. Neste caso o usuário deverá recarregar o CCP após o término de sua execução.

Figura 2. A imagem de memória do CP/M



Fonte: autor, 1987.

1.3.2 A página 0 (zero)

Após sua carga o CP/M ocupa uma área de aproximadamente 7 Kbytes no topo do espaço do endereçamento. A área correspondente aos primeiros 256 bytes denominada “página 0” é reservada para conter diversos parâmetros do sistema, vetores de entrada para os diversos componentes do CP/M e áreas para dados temporários.

A tabela 1. apresenta o endereço e o conteúdo dos componentes da página 0.

Tabela 1. Conteúdo da página 0

Posição de memória	Função
0000 – 0001	vetor de partida "morna"
0003	IOBYTE (parâmetro para rotinas de E/S)
0004	unidade de disco corrente
0005 - 0006	vetor de entrada para FDOS
0008 - 005B	áreas para interrupção e dados temporários
005C - 007C	bloco de controle de arquivos
007D - 0D7F	bloco de controle para acesso randômico
0080 - 00FF	buffer para disco

Fonte: autor, 1987.

1.4. Como opera o CP/M

Durante a operação o CCP apresenta o “prompt” e fica esperando que o usuário digite um comando. Na realização destas tarefas o CCP usa o BIOS para executar as operações de E/S. Uma vez digitado o comando o BIOS coloca-o buffer do CCP.

Com relação aos comandos existem os comandos residentes e os não residentes. Comandos residentes são aqueles contidos junto com o sistema. No caso de ser digitado um comando residente este é imediatamente executado. Se o comando for do tipo não residente o CCP considera que o usuário refere-se a um programa a ser carregado na TPA. Este programa deverá estar na unidade de disco corrente na forma de um arquivo cujo nome será o digitado concatenado ao sufixo “COM”. Assim sendo o CCP encarrega o BDOS de ler o arquivo do disco e carregá-lo na TPA para sua execução. No caso do nome digitado não ser um programa residente em um arquivo do disco corrente o CCP envia através do BIOS uma mensagem de erro.

Desta forma os programa criados pelo usuário serão executados como se fossem comandos não residentes bastando que estejam na unidade de disco corrente e possuam o sufixo “COM”.

1.5. MP/M

A Digital Research desenvolveu o MP/M (Multi Programing Monitor for Microcomputer) para aplicações multiusuário, tempo compartilhado em sistemas multiprogramados e até mesmo para aplicações em tempo real. Este sistema é baseado em um núcleo que suporta a multiprogramação e em um gerenciador de arquivos compatível com o CP/M.

Sendo a parte principal do sistema o núcleo suporta:

- multiprogramação
- gerencia de filas
- gerencia de flags
- gerencia de memória
- temporização

Por se tratar de um sistema multiprogramado as filas são utilizadas para:

- comunicação entre tarefas
- sincronização entre tarefas
- exclusão mútua

Os flags permitem a ocorrência de TRAPS (interrupções de software) para o atendimento de serviços.

O gerenciamento de memória visa contornar uma limitação dos micros de 8 bits com relação ao pequeno espaço de endereçamento (8 bits). Desta forma o MP/M permite o uso de até 8 bancos de memória com 48 kbytes cada. Através do chaveamento dos diversos bancos é possível aumentar a memória disponível.

1.5.1 A estrutura do MP/M

O MP/M possui uma estrutura similar a do CP/M consistindo nos seguintes componentes:

- XIOS (BIOS com E/S estendida)
- XDOS (BDOS com o sistema operacional de disco estendido)
- CLI (Comand line interpreter)
- TMP (Terminal message processador)

1.5.2. Limitações do MP/M para aplicações em tempo real

A limitação do MP/M deve-se ao fato da segmentação da memória ser fixa, sendo definida durante a geração do sistema. Desta forma ocorrem dois grandes problemas:

a) O número de tarefas residentes na memória fica limitado ao número de segmentos definidos. Um grande número de segmentos implica em reduzir grandemente o tamanho das tarefas. Por outro lado, poucos segmentos implicam em poucas tarefas residentes o que torna o sistema muito lento devido a necessidade do “SWAP”.

b) A imposição de definir os segmentos durante a geração do sistema implica, como já foi mencionado acima, num compromisso entre tamanho das tarefas e número de tarefas residentes (na memória). Fatalmente tal compromisso leva a uma má utilização de memória devido a ocorrência da fragmentação interna aos segmentos.

APÊNDICE 2
O COMPILADOR PASCAL/Z

1.1. Introdução

O compilador Pascal/Z da Ithaca Intersystems Inc. foi desenvolvido para compilar programas escritos na linguagem Pascal para macro-código Z80 sendo bastante semelhante a definição do Jensen and Wirth's Pascal User Manual and Report (Segunda Edição). Atualmente é disponível para rodar sob o sistema operacional CP/M da Digital Research's.

O pacote de software do Pascal/Z inclui o código objeto para ambas versões dos compiladores, 48K e 54K, o montador (ASMBLE/Z), o ligador (LINK/Z), e os objetos e fontes comentados para as rotinas da biblioteca. A documentação que se dispõe inclui os manuais ASMBL/Z e LINK/Z além do Pascal/Z User's Manual.

2.2. Objetivos do Pascal/Z

Pascal/Z foi desenvolvido para ser utilizado numa grande variedade de aplicações constituindo-se num compilador descendente recursivo para o microprocessador Z80. Seus objetivos de projetos são:

- rodar em CPU Z80;
- gerar código reentrável que possa ser gravado em memória ROM;
- criar um compilador que possa ser facilmente reinstalado para uso como cross-compiler;
- ser escrito em Pascal para fácil manutenção e boa confiabilidade;
- ser eficiente quanto a quantidade de código gerado;
- acrescentar extensões para aplicações científicas e industriais.

2.3. Características da implementação

- O código gerado é reentrável podendo ser gravado em ROM;
- Programas dedicados podem ser tão pequenos quanto algumas centenas de bytes;
- O compilador gera código otimizado para Z80;
- Gera código macro-assembler para o Z80 podendo incluir as linhas da fonte como comentários;
- suporta compilação em separado dos programas do usuário.

2.4. Extensões da linguagem padrão

- Compilação em separado;
- Permite interface com rotinas escritas em Assembly como rotinas externas;
- Inclusão de arquivos através de diretiva INCLUDE;
- Suporta o tipo STRING;
- Suporta o acesso direto a arquivos (acesso randômico);
- Possui cláusula ELSE no comando CASE;
- Entrada e saída simbólica para tipos de dados enumeração;
- Constantes inteiras podem usar os operadores +, -, *, DIV;
- Funções podem retornar tipos de dados estruturados;
- Permite otimização do tipo TEMPO/ESPAÇO quando usa o comando CASE;

2.4.1. Restrições do Pascal/Z

- Não implementa E/S padrão através de GET/PUT;
- Não implementa parâmetros procedurais. Uma função ou procedure não pode ser passada como parâmetro;

- Armazenamento dinâmico implementado através de NEW, MARK e RELEASE ao invés de NEW e DISPOSE;
- Não permite acessar posições absolutas de memória;
- Não possui tipos de dados BYTE e WORD.

2.4.2. Sistema Requerido

O Pascal/Z roda sob o sistema operacional CP/M da Digital Research sendo recomendado a versão 2.2 ou superior pois versões anteriores não suportam acesso randômico a arquivos (chamado de Acesso Direto a Arquivos no Pascal/Z).

Quando a compilação está em andamento são necessários 48K de RAM para o compilador (54K para versão com tabelas de símbolos/tipos expandidas), mais a memória necessária para o sistema operacional (em geral 8K).

A configuração recomendada consiste em 64K de memória com duas unidades de disquete.

2.4.3. Como rodar o Pascal/Z

Para rodar o compilador é necessário que o arquivo PAS2 esteja na unidade de disquete corrente devendo-se digitar:

PASCAL <nome>. <unidade fonte> < unidade saída> < unidade listagem>

Onde:

<nome> é o nome do arquivo fonte com extensão .PAS;

<unidade fonte> letra da unidade onde está a fonte;

< unidade saída> letra da unidade onde está gravado o macro-código Z80 gerado pelo compilador;

< unidade listagem> letra da unidade onde está gravado a listagem gerada pelo compilador.

Um espaço em qualquer local indica a unidade de disquete corrente.

2.4.4 Exemplos

A) PASCAL PRIMES.BC

Compila a fonte PRIMES.PAS da unidade B, gravando o macro-código PRIMES.SRC na unidade C e a listagem PRIMES.LST na unidade A.

Após a compilação o programa deve ser montado e ligado antes de poder ser executado.

Para montar o programa digite:

A) ASMBL MAIN, PRIMES/REL

Este comando faz com que o montador ASMBLE/Z automaticamente procure pelos arquivos .SRC associados com os nomes MAIN e PRIMES. MAIN.SRC é o arquivo que contém definições e macros que devem ser montadas com a saída do compilador. Deve sempre ser o primeiro arquivo a ser especificado.

Após a montagem é preciso fazer a ligação do programa com as sub-rotinas necessárias digitando:

A) LINK PRIMES/N:PRIMES/E

Este comando faz com que o ligador LINK/Z faça a ligação do programa com as rotinas no arquivo LIB.REL. Somente são ligadas as rotinas usadas pelo programa permitindo uma redução do código gerado.

Para execução do programa basta digitar:

A) PRIMES

Para maiores informações devem ser consultados os manuais Pascal/Z User's Manual, ASMBLE/Z e LINK/Z indicados na bibliografia.

2.4.5 Rotinas externas

Rotinas externas são tratadas internamente exatamente como procedures e funções do Pascal. Estas rotinas são declaradas de modo semelhante a declaração FORWARD, trocando esta palavra reserva por EXTERNAL. Por sua vez os módulos externos devem ser montados usando o arquivo EMAIN.SRC enquanto que o programa principal deve continuar a usar a MAIN.SRC.

Para correta utilização das rotinas externas devem ser respeitadas as seguintes convenções:

- Os registradores X, Y e os pares alternativos BC', DE', HL' não podem ser alterados;

- Quando retornar de uma rotina externa o acumulador deve estar zerado;

- Se a rotina for uma função então:

a) BOOLEANS retornam CARRY em um → VERDADEIRO

CARRY zerado →> FALSO

b) Valores escalares não REAL retornam o valor no par DE;

c) REALs retornam nos quatro bytes em cima dos parâmetros de função na pilha;

d) Tipos de dados estruturados retornam nos bytes em cima da lista de parâmetros na pilha (onde n= tamanho do valor retornado).

- Toda rotina externa é responsável pela remoção de todos os seus parâmetros da pilha antes de retornar.

Todos os parâmetros para a rotina externa são passados através da pilha, sendo empilhados na ordem na qual são declarados. Após todos parâmetros estarem na pilha a rotina externa é chamada através da instrução CALL do Z80.

No arquivo EMAIN.SRC existem duas macros, ENTR e EXIT, que podem ser usadas para simplificar a construção das rotinas externas. ENTR deve ser usada da seguinte forma:

ENTR D, LVL, VSIZ

Onde D é um argumento fictício (pode ser usado qualquer valor), LVL indica o nível de rotina (o nível global é 1, para rotinas externas deve ser usado o valor 2) e VSIZ é o número de bytes necessários para variáveis locais da rotina externa.

Assumindo ser n o número de bytes dos parâmetros para a rotina externa então estes podem ser acessados da seguinte maneira:

n+7 (IX); primeiro byte dos parâmetros = 8+n-1 (IX)

n+6 (IX); segundo byte dos parâmetros = 8+n-2 (IX)

8(IX); último byte dos parâmetros = 8+n-n (IX)

A área de dados locais (requisitada por VSIZ) é endereçada:

0(IX); primeiro byte da área local
1-VSIZ(IX); último byte da área local

Para funções não reais o valor de retorno (inicializado com zero por ENTR) é armazenado em:

2(IX); byte inferior do valor retornado
3(IX); byte superior do valor retornado

Funções reais tem o valor armazenado em:

n+8(IX); byte menos significativo

.

.
.
.

n+11(IX); byte mais significativo (expoente)

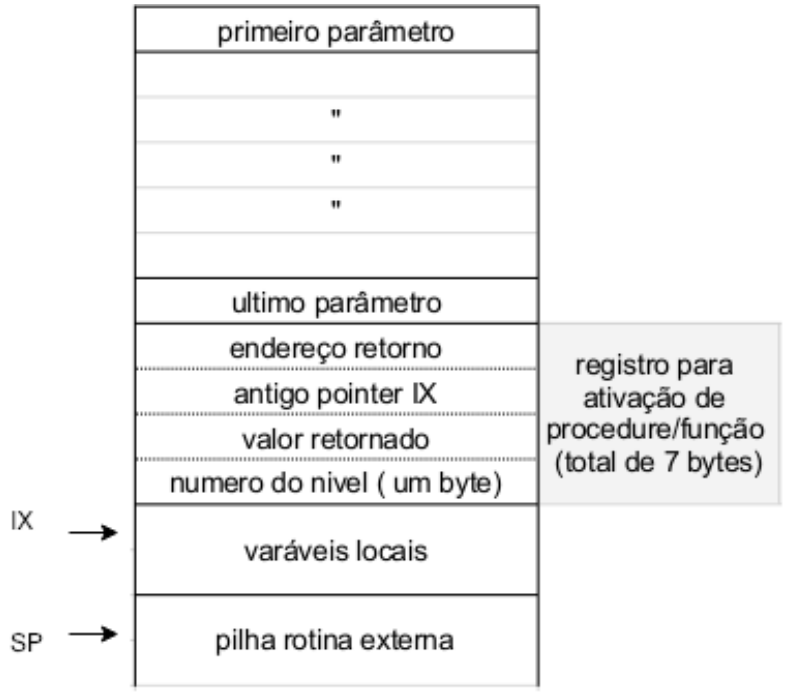
Funções com valor de retorno estruturado tem esse valor gravado em:

n+8(IX); byte menos significativo

n+tamanho-1(IX); byte mais significativo

Na tabela 2, Após a execução da macro ENTR a pilha fica da seguinte maneira:

Tabela 2. Após a execução da macro ENTR



Fonte: autor, 1987

EXIT é usado da forma:

EXIT D, PARMSZ

Onde D é o mesmo argumento fictício e PARMSZ é o número de bytes dos parâmetros passados para a rotina externa. Esta macro remove a área de dados locais, restaura a pilha e o registrador IX para valores originais, limpa o acumulador, coloca o valor de retorno correto e retorna para a rotina que chamou a rotina externa.

2.5. Organização da pilha

Todo código gerado pelo Pascal/Z pode ser gravado em memórias ROM sendo também reentrável pois cada programa (tarefa) possui sua própria pilha, tal como é descrito a seguir.

Os programas em Pascal/Z começa pela inicialização de suas pilhas através do macro ENTR cujo código para expansão se encontra no arquivo MAIN.SRC. Todas variáveis não dinâmica são armazenadas na pilha. Pela alteração do código de ENTR é possível que várias tarefas compartilhem a mesma imagem do código de um programa em Pascal/Z tendo cada uma sua própria pilha.

Durante a execução de um programa os registradores IX e IY do Z80 (ponteiros) apontam para as seguintes áreas de dados:

IY → área de dados globais
IX → área de dados locais

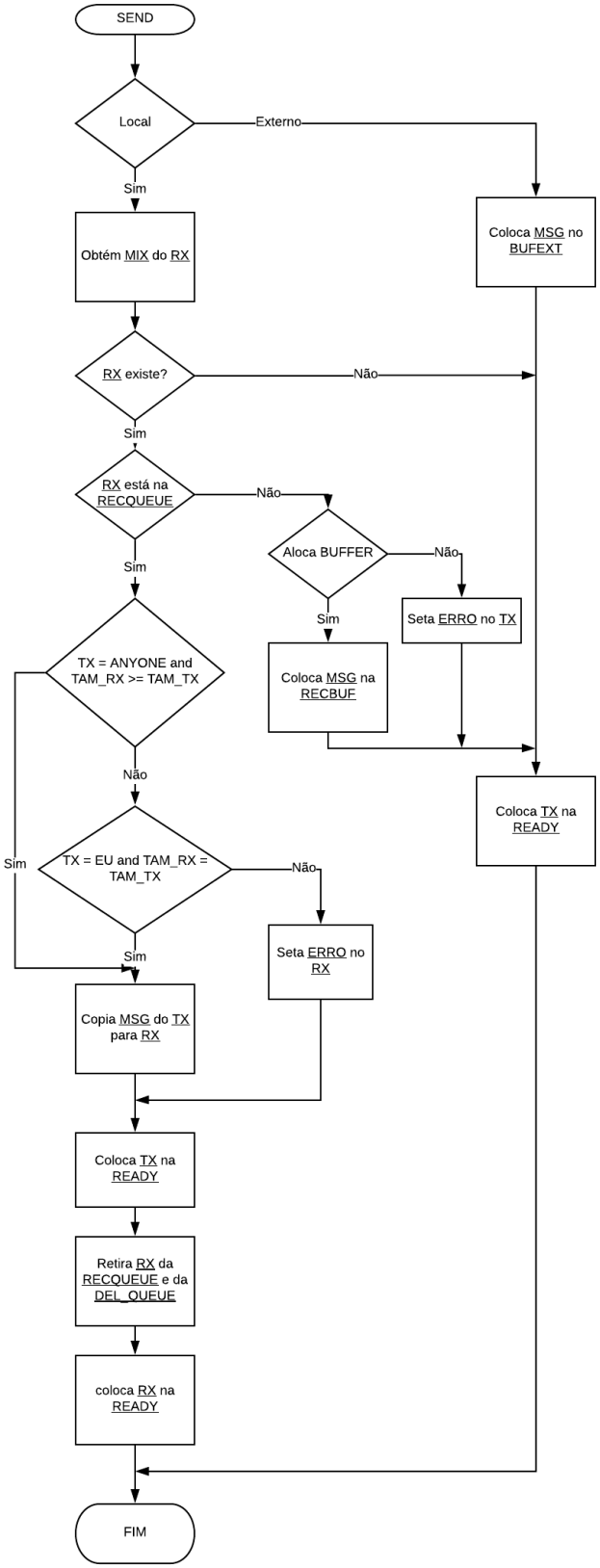
Como todas as variáveis são armazenadas e acessadas na pilha esta não pode ser relocada após sua inicialização pois visando uma melhor eficiência todas referências são feitas por endereço absoluto e não relativo.

O heap como o stack é alocado em tempo de execução e não em tempo de compilação. Isto significa que o heap também pode ser inicializado em um ponto diferente da memória para cada tarefa. O valor inicial do heap é dado pelo identificador LAST, sendo este valor igual a próxima posição de memória após o programa do usuário.

APÊNDICE 3

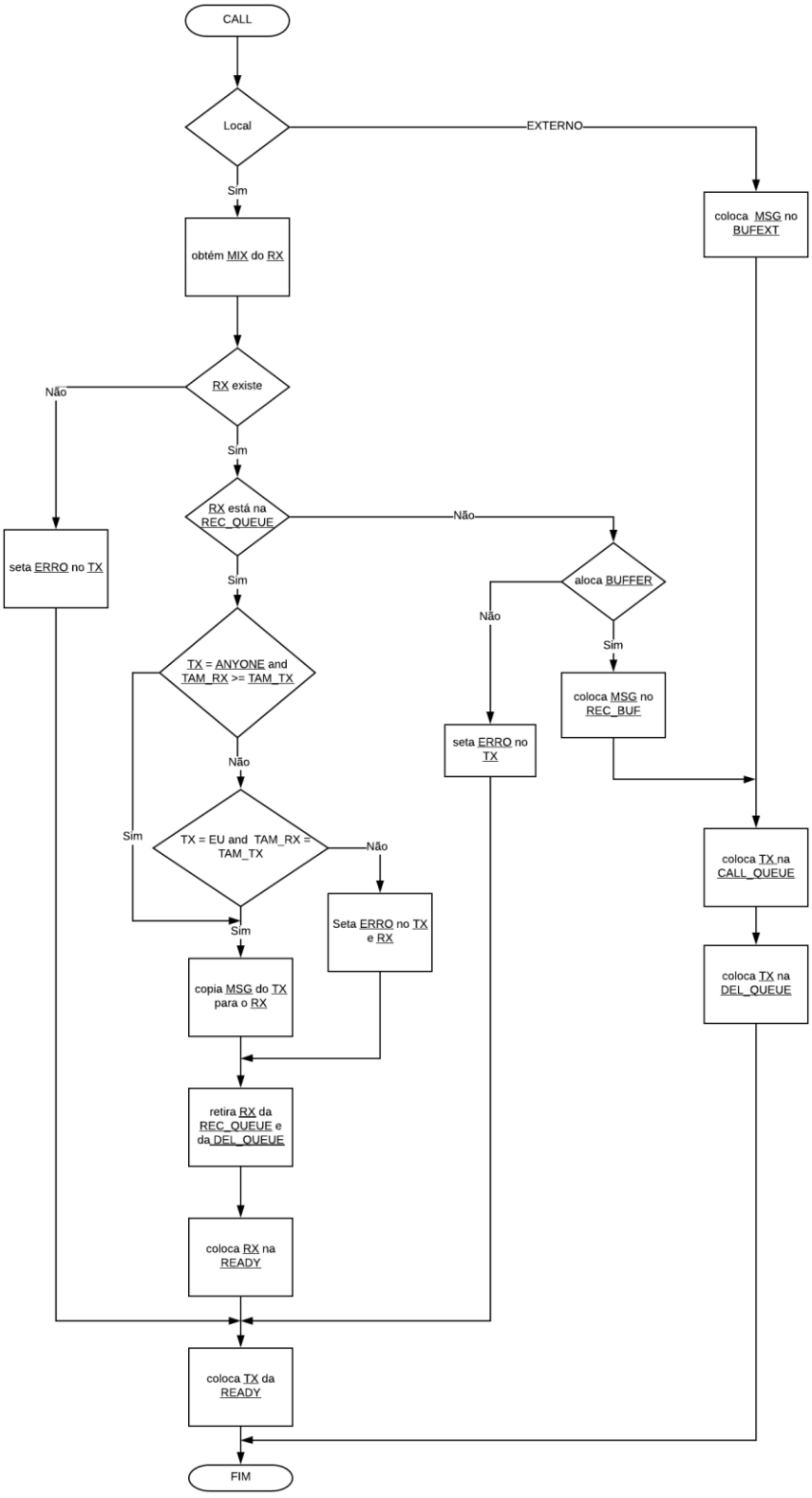
FLUXOGRAMA DAS DIRETIVAS

Ilustração 1. Diretiva SEND



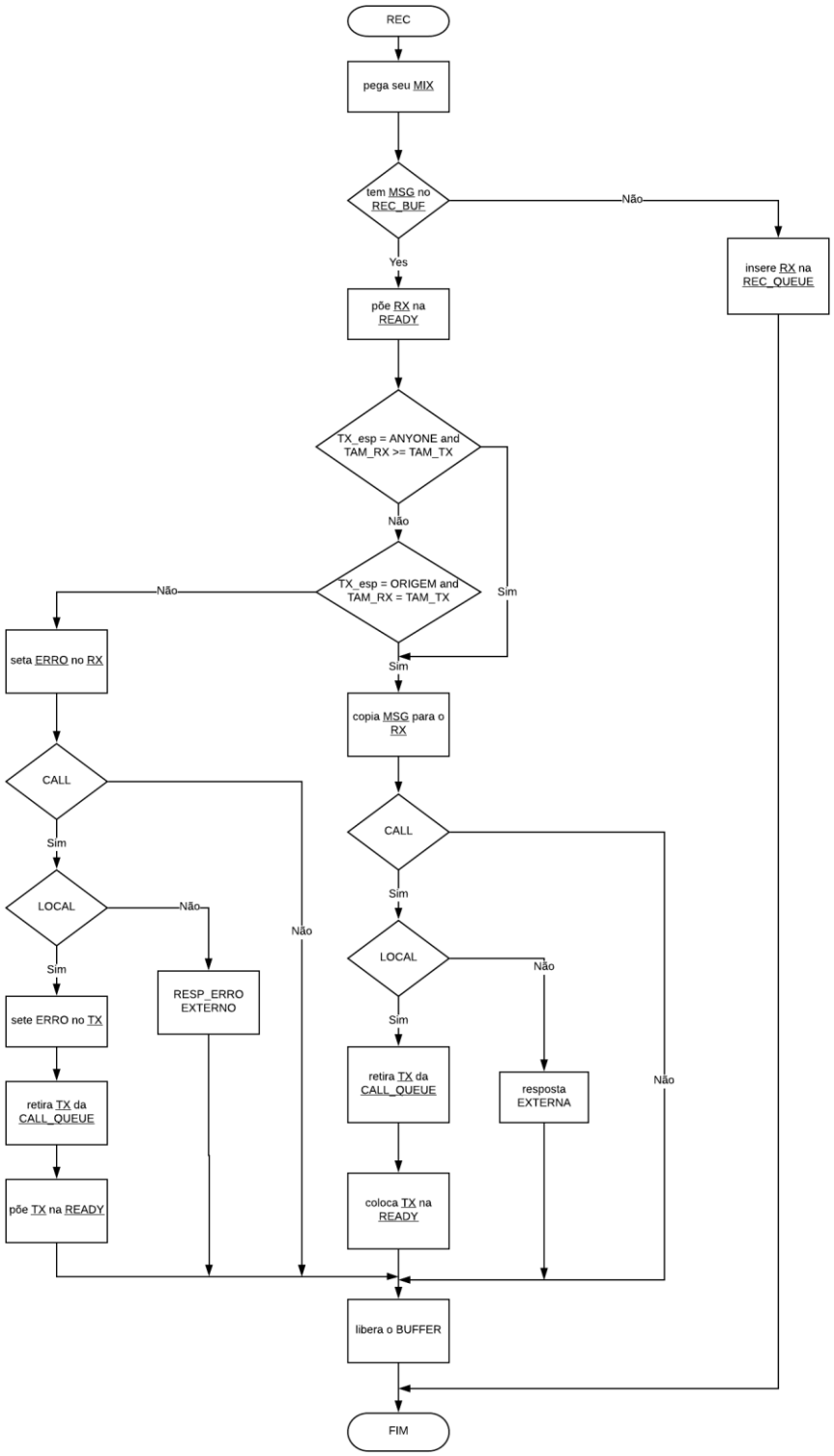
Fonte: autor, 1987

Ilustração 2. Diretiva CALL



Fonte: autor, 1987.

Ilustração 3. Diretiva REC



Fonte: autor, 1987