UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

DEPARTAMENTO DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

# Um modelo para configuração de núcleos para tempo real

*autor:* Maria Cristina Amim Zabeu

*orientador:* Maurício Ferreira Magalhães

Tese apresentada à Faculdade de Engenharia Elétrica
da Universidade Estadual de Campinas,
como parte dos requisitos exigidos
para a obtenção do título de mestre em
Engenharia Elétrica.

Este exemplar corresponde à redação final da tese defendida por Maria Cristina Amim Zabeu e aprovada pela Comissão Julgadora em 15/03/90

# Um modelo para configuração de núcleos para tempo real

Maria Cristina Amim Zabeu

20 de novembro de 1989

Obrigada a todos que,
direta ou indiretamente,
influiram neste trabalho.

## Sumário

Este trabalho propõe um modelo para o desenvolvimento de núcleos de tempo real de forma a configurá-los de acordo com as necessidades das aplicações que os utilizam. A configuração do núcleo pode ser feita com o uso de um sistema baseado em tabelas, ou então com um sistema baseado em conhecimento (sistema especialista); ambos são delineados neste trabalho. São discutidos também uma implementação de um núcleo de acordo com o modelo proposto e uma particularização para uma dada aplicação (implementação de protocolos).

## Abstract

A model for the development of real-time kernels is proposed, in order to make feasible their configuration to meet application needs. This configuration is discussed and two methods are given, one based on information tables and the other based in knowledge-based systems. A prototype implementation is also discuted, as well as the kernel specification to an application (protocol implementation).

# Conteúdo

# Capítulo 1

# Introdução

O objetivo deste trabalho é a proposta de um modelo de desenvolvimento de software básico (núcleo), a partir do qual seja possível efetuar configurações, adequando-o a diferentes necessidades. A motivação para a construção de um núcleo que seja adaptável a "alto" nível surge quando se verifica a diversidade de aplicações que necessitam deste tipo de software básico, com requisitos de operação e suporte hardware diferentes entre si. A configuração proposta neste trabalho encara um núcleo como um conjunto de unidades computacionais (*entidades*), que por sua vez são formadas de tratadores e fornecedores de serviço (*gerências*). Com estes elementos é possível efetuar tanto a construção funcional quanto a escolha de valores de uso "interno" do núcleo. Para a configuração são delineadas duas ferramentas: uma baseada em tabelas, e outra baseada em um sistema especialista.

Um pequeno núcleo foi implementado segundo o modelo, bem como um protótipo de um configurador. A aplicação deste modelo em uma aplicação particular - implementação de protocolos - foi discutida, mostrando que é possível a particularização de um núcleo para um determinado fim, mantendo a estrutura básica fornecida pelo modelo.

Este trabalho está dividido nas seguintes partes:

- Capítulo 2: apresenta um estudo resumido de diversas abordagens utilizadas para desenvolvimento de software e propõe um modelo para a construção de um núcleo.

- Capítulo 3: discute características e apresenta aspectos de implementação para a implementação de um núcleo segundo o modelo proposto no capítulo anterior.

- Capítulo 4: apresenta a motivação para configuração de núcleos, e discute dois tipos de configuradores.

- Capítulo 5: para o núcleo implementado segundo o modelo proposto,

são exemplificadas sua utilização e possíveis modificações. São criticadas também as opções de implementação utilizadas.

- Capítulo 6: o modelo proposto é utilizado na discussão da particularização de um núcleo a uma aplicação - implementação de protocolos, a partir das características específicas desta.

Nos anexos são apresentados os pseudo-códigos do núcleo implementado e as interfaces funcionais aos serviços fornecidos por este núcleo.

# Capítulo 2

# Modelo de formação de núcleos

## 2.1 Software básico

O software básico é o conjunto de código que faz a interface entre um hardware e uma aplicação. O software básico, em geral, gerencia diversas atividades que utilizam o processador e o acesso a dispositivos periféricos que compõem o sistema [1].

Quando as operações da aplicação envolvem o fator "tempo real" [2], [3], [4], isto é, têm compromissos com o mundo exterior no tocante a tempos de execução e de resposta a estímulos, exigem-se características particulares para o conjunto hardware/software básico/aplicação [5], [6], [7], [8], [9].

Nos primeiros sistemas de controle em automação o software aplicativo era implementado diretamente sobre o hardware. Com a complexidade crescente do hardware surgiu a necessidade do fornecimento de uma máquina virtual, sobre a qual a aplicação seria implementada. Este conceito deu origem à idéia de "núcleo" [10], [11]. Com o uso de um núcleo em um sistema de controle têm-se:

- independência da aplicação quanto a detalhes da arquitetura hardware
- facilidade de adição de novas operações
- a possibilidade de melhor utilização dos recursos da máquina hospedeira.

## 2.2 Características de um núcleo

Um núcleo possibilita utilizar a máquina de forma otimizada. Isto é obtido trabalhando com um ambiente multi-tarefa, ou seja, onde o processador nunca fica ocioso. Mesmo quando alguma operação deve esperar por algum "evento" e portanto se torna inativa, existe um controle sobre as ações a

serem executadas de forma a escolher outra atividade para ocupação do processador. É o núcleo o responsável por este controle de ocupação de processador; existem características associadas a este controle que determinam a maneira mais adequada de faze-lo tendo em vista as necessidades de um dado "usuário". Estas características podem ser, por exemplo, prioridades e privilégios associados às ações que devem ser executadas. Mecanismos de garantia da integridade de cada ação que ocupa o processador também são providos pelo núcleo. O núcleo oferece meios para a aplicação definir quais são as atividades que vão ocupar o processador e quais as características associadas a elas. Ao núcleo é reservado o controle dos recursos da máquina, no que concerce à ocupação, liberação e uso. Também é função do núcleo controlar os meios pelos quais as diversas atividades que compõem a aplicação se comunicam e sincronizam.

Um núcleo caracteriza-se por:

- controlar os recursos da máquina hospedeira
- fornecer meios através dos quais uma aplicação divide sua operação em atividades (tarefas) que possam ser disjuntamente executadas
- controlar a execução destas diversas atividades
- fornecer e controlar meios para que as atividades se coordenem e se comuniquem

Estas características se refletem, na implementação de um núcleo, nas diversas funções por ele executadas.

## 2.3 Elaboração de um núcleo

Um núcleo pode ser construído de várias formas, sempre englobando as características abordadas no ítem anterior. As diversas funções por ele

executadas podem ser agrupadas de forma a facilitar a programação, para adequar-se a uma dada aplicação, ou para permitir atualizações e alterações de forma menos drásticas. A engenharia de software [12], [13], [14], [15], [16], [17], [18] e a teoria de sistemas operacionais [19], [20], [21], [22], mostram que existem diversas visões para a construção de software que podem auxiliar na elaboração de um núcleo.

Apresenta-se a seguir um estudo enfatizando algums modelos; algumas de suas particularidades não estarão suficientemente relatadas, uma vez que o objetivo é analisar características aplicáveis ao desenvolvimento de um núcleo.

### 2.3.1 Abordagem "Programming-in-the-large" / "Programming- in-the-small"

Esta forma de elaboração sugere que existam duas visões para a construção de um mesmo software:

- uma visão em que as funções do software sejam tratadas a nível geral
- uma visão que considere a forma de trabalho de cada função.

A um nível "macroscópico" ("large") o software é composto de elementos que definem as suas características; desta maneira, a presença ou acréscimo de funções é ligada a uma necessidade funcional da aplicação que vai utilizá-lo.

Ao nível "microscópico" ("small") entra-se no mérito da implementação de cada função, no algorítmo ou estrutura de dados utilizada. A programação pode ser feita e testada, ao nível microscópico, de forma genérica e não necessariamente dependente do ambiente onde será utilizada.

Em um núcleo, esta visão de construção permite a exploração de diversas formas de implementação de uma função. Pode-se obter com isso

funções adequadas a uma aplicação no sentido de tamanho de código ou desempenho. A construção de funções pode ser feita utilizando-se outras já implementadas.

A partir destas duas "visões" o núcleo pode ser elaborado tanto a um "alto" nível, somente no tocante à funcionalidade, ou então ser visto a um nível de detalhes. O software aplicativo ganha com isso maior "independência", deixando aspectos particulares do software básico separados do contexto da aplicação.

A possibilidade de configuração de conjuntos de funcionalidades, gerando "núcleos" diferentes, é uma alternativa possível utilizando este modelo.

### 2.3.2 Abordagem por objetos

O modelo orientado a objetos [23], [24] enfatiza a caracterização dos componentes de um sistema em elementos a partir de suas propriedades invariantes e de seu comportamento. Estas propriedades são preservadas por um conjunto de operações que compõem a forma pela qual os objetos são manipulados. Existem operações específicas sobre um objeto para alterar ou determinar o estado em que o mesmo se encontra. Obtém-se com isso um encapsulamento no que diz respeito ao objeto frente à aplicação que o utiliza, uma vez que o comportamento do mesmo se define por meio das operações que agem sobre ele e não por ação direta. Observa-se que, se estas funções forem parametrizadas, elas podem ser aplicadas a vários objetos. Neste contexto, objetos são do mesmo "tipo" quando partilham o mesmo conjunto de operações.

No escopo de um núcleo, os vários serviços fornecidos podem ser encarados como objetos. Uma composição possível liga cada objeto a um recurso da máquina; outra liga um objeto a uma funcionalidade. Objetos podem ser compostos para formar novos objetos, e assim se obtém tanto uma cons-

trução hierárquica como um fator de reusabilidade. Um mesmo tipo de objeto pode controlar diversos recursos da máquina, alterando-se apenas os parâmetros da operações que agem sobre ele; um novo objeto pode ser acrescentado para operar sobre uma nova potencialidade da máquina. A visão do usuário pode ser a de um único objeto que tem as características de um núcleo.

Pontos importantes a ressaltar nesta abordagem são a auto-suficiência de um objeto, sua caracterização e encapsulamento por meio das operações que agem sobre ele e a viabilidade de composição hierárquica sem afetar a estrutura interna de um objeto e sem efeitos colaterais para o usuário final.

### 2.3.3 Abordagem em camadas hierárquicas

Um outro modelo de construção de software sugere que a partir da máquina - puramente hardware - se construam "camadas" até que se forneça um ambiente único para a aplicação. Cada uma dessas camadas proporciona um nível maior de abstração, até que se tenha uma visão de uma máquina estendida.

Isola-se, geralmente, em cada camada, uma certa funcionalidade, o que permite controlar a complexidade do software e facilitar sua construção e teste.

Cada camada é um elemento isolado que utiliza funções da camada inferior e fornece funções à camada superior, mantendo assim uma estrutura hierarquizada.

O modelo usual é o da colocação no nível mais baixo das funções que serão usadas de forma genérica pelas outras camadas. No caso de um núcleo, as camadas seriam responsáveis, na ordem hierárquica, pelo:

- o controle da máquina,
- o controle de atividades sobre a máquina,

- o controle de memória

e sucessivamente. Geralmente orienta-se pelo tipo de aplicação a ser satisfeita para a estruturação das camadas e localização das funcionalidades entre camadas.

Uma característica de um núcleo construído segundo este modelo é o fornecimento de uma "máquina virtual de execução" a cada camada, podendo uma determinada aplicação optar pela que mais lhe convém, seja por um fator de otimização, seja por simplificação, ou mesmo acrescentando funções na forma de "novas camadas".

### 2.3.4 Abordagem em anel

Esta abordagem pode ser tomada como uma variante do modelo hierárquico. Neste caso existe um conjunto central de operações sobre o qual executam as diversas funções fornecidas à aplicação. Este conjunto central oferece os serviços mínimos a serem utilizados pelos prestadores de serviço à aplicação. A diferença fundamental está no fato de que as funções se encontram dentro de um mesmo nível hierárquico, sobre um mesmo "ambiente" fornecido pelo conjunto central.

### 2.3.5 Considerações sobre as abordagens apresentadas

Todos os modelos apresentados têm características interessantes do ponto de vista de construção de um sistema (hardware+aplicação); todos tentam separar os detalhes de implementação do contexto da aplicação, permitindo ao usuário do software se ocupar apenas do aspecto funcional.

No modelo microscópico/macroscópico é enfatizada a separação entre funcionalidade e construção de software. Não se preveem ligações de

dependência entre funcionalidades; a programação é totalmente independente do aspecto funcional do software. O objetivo principal do modelo microscópico/macroscópico apresentado é separar características que concernem apenas à implementação de um software sobre uma determinada máquina, daquelas que estão ligadas à construção ou configuração de um ambiente a partir de um conjunto de serviços. Isto permite a programação modular e independente dos elementos que compõem um sistema e uma posterior configuração destes, resultando em ambientes finais distintos. Um exemplo do uso desta abordagem existe no projeto CONIC [25].

No modelo por objetos, há possibilidade do mapeamento dos elementos do software em elementos computacionais, isolando o aspecto de implementação mas liberando a relação funcional entre objetos. Ou seja, a abordagem por objetos procura mapear entidades do mundo real e seu inter-relacionamento em estruturas computacionais, quando geralmente se faz o inverso (adequa-se o problema às ferramentas disponíveis para resolvê-lo). Ao encarar cada recurso de um ambiente como um objeto, um núcleo é simplesmente um outro objeto, composto por eles, e este, por sua vez, é parte componente da aplicação. A funcionalidade, neste caso, é provida pelas operações que se fazem sobre os objetos. Dentro da área de sistemas operacionais [1] existem vários casos onde a modelagem por objetos é utilizada, entretanto o grau de fidelidade com o qual a mesma é aplicada varia bastante [26], [23].

No modelo hierárquico é imprescindível saber quais as funções da camada inferior e quais devem ser fornecidas à camada superior. O modo de implementação não é relevante, mas a estrutura funcional é mais rígida que a prevista nos demais modelos. A abordagem hierárquica é bastante difundida e vem sendo usada em diversos sistemas operacionais, como o 370 OS/MVT da IBM [21] [2].

[1] uma vez que o modelo orientado a objetos tem uma ampla gama de aplicações

[2] inclui-se neste comentário também o modelo em anel, tendo como exemplo o RMX-80

O esquema hierárquico é conceitualmente englobado pelo modelo de objetos, embora aqui não se tenha um encapsulamento funcional único, pois um serviço é fornecido por operações de diversas camadas.

Observa-se que em todos os modelos apresentados buscam-se as seguintes características:

- uma possibilidade de composição funcional de acordo com necessidades da aplicação;
- uma separação da programação propriamente dita do ambiente funcional fornecido;
- a utilização de elementos computacionais comuns;
- a estruturação isolando dependências do ambiente físico e também apropriada para incorporar novas facilidades ou modificar serviços já presentes.

Do ponto de vista de um núcleo a facilidade de se ter a composição de diversas funções justifica-se frente à diversidade de requisitos das aplicações. A reusabilidade de elementos de software busca obter maior produtividade na programação, ganhando-se também na questão de custos, uma vez que se economiza tempo tanto em implementação quanto em testes. A portabilidade entre vários ambientes é desejada pelo aspecto de reaproveitamento de trabalho e diminuição de custos. A expansibilidade é interessante já que sistemas de controle têm uma longa vida útil e normalmente são acrescidos de novos recursos de hardware neste período.

[27]

## 2.4 Uma proposta de modelo para construção de um núcleo

Utilizando as idéias apresentadas nos ítens anteriores, e procurando manter como objetivos portabilidade, reusabilidade, expansibilidade e configurabilidade, será descrita neste ítem uma proposta de modelo para a construção de um núcleo. Este enfoque busca adequar conceitos das várias abordagens de construção de software discutidas, de forma a utilizar suas melhores características para a elaboração de um "núcleo".

O modelo a ser descrito baseia-se no conceito de "entidade". Uma entidade é definida como um conjunto de funções e serviços que refletem características que um núcleo deve possuir. Como estas características podem variar de acordo com os recursos disponíveis e as necessidades de uma aplicação, três tipos de entidades podem existir:

- a entidade essencial
- a entidade básica
- a entidade de serviço à aplicação

Uma entidade fornece "funções"; uma função é controlada por uma "gerência". Uma gerência engloba os códigos ligados a operações ("serviços") sobre elementos, lógicos ou físicos, do ambiente computacional, assim como à composição destas operações no fornecimento de funções.

Assim posto, uma entidade é formada de um número variado de gerências, que por sua vez englobam serviços, sendo que a nível externo são vistas apenas funções.

A figura 2.1 - pag. 23 mostra um exemplo da relação entidade/gerência/serviço/função, onde

- $F_i$: função fornecida pela entidade

- $G_i$: gerências da entidade
- $S_i$: serviços
- $E_i$: entidade

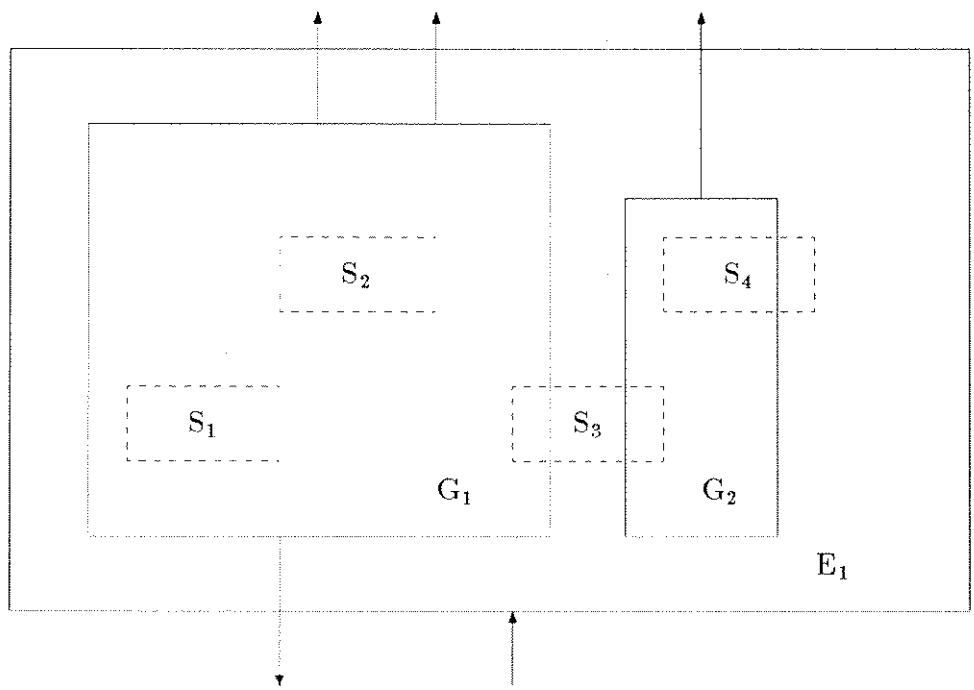



Figura 2.1: Relacionamento entre elementos de entidades

A seguir será apresentada a conceituação de cada uma das entidades para a construção de núcleo segundo este modelo.

### 2.4.1 Entidade Essencial

A entidade essencial fornece as funções que caracterizam minimamente um núcleo. Nesta entidade estão presentes as gerências ligadas ao hardware da máquina hospedeira, ao fornecimento de meios para definir e controlar unidades de execução da aplicação (tarefas), e à elaboração e coordenação de elementos de comunicação e sincronização entre estas. Alguns serviços desta entidade podem ser disponíveis à aplicação.

A gerência ligada ao hardware engloba basicamente a programação e controle de dispositivos que controlam recursos físicos, inclusive relógios, da máquina. É a parte naturalmente mais dependente das características da máquina, e portanto menos "portátil".

A gerência ligada à definição e controle de unidades de execução, ou *tarefas*, partilha usualmente de serviços da gerência de controle hardware, uma vez que é responsável pela alteração de contextos de execução que se dão por estímulos de tempo e por estímulos externos. Entretanto, a parte lógica, como a que engloba verificação de prioridades ou privilégios, é genérica e portanto independente da máquina.

A gerência que provê comunicação e sincronização entre tarefas utiliza diferentes estratégias para transferência de informação; eventualmente serviços sobre recursos do hardware (por exemplo, conexão de interrupções à transferência de informação), são incorporados à gerência.

### 2.4.2 Entidade Básica

A entidade básica caracteriza-se por fornecer funções que podem ser utilizadas pelas outras entidades ou mesmo pela própria aplicação, mas que

têm um perfil genérico. Ou seja, o conceito de "básico" vem da tentativa de se unificar o tratamento de um problema particular de forma a que esta solução possa ser usada em vários contextos. Entretanto, um serviço da entidade básica pode estar inserido na implementação de outra entidade, ou então, a forma de implementação para o fornecimento de uma mesma função pode ser diferente para contextos distintos. Neste caso, a solução do problema é duplicada, sendo implementada na entidade que necessita uma abordagem particular, e de forma genérica na entidade básica.

Situam-se nesta entidade funções como a de manipulação de filas, de interface a recursos de entrada/saída, de controle de memória e de manutenção de calendário. Por exemplo, para o caso de manipulação de filas, tem-se que:

- é uma função que é usada tanto pela aplicação quanto pela entidade essencial, no controle de execução ou sincronização de tarefas;
- podem existir diversas formas de controle de filas
- pode ser implementada pelos usuários.

o que caracteriza seu perfil como entidade básica.

A interface para E/S encontra-se classificada neste ítem, contudo inclui funções ligadas à entidade essencial para acesso às particularidades da máquina. Nesta classificação sugere-se que esta interface seja mais lógica, independente da máquina física.

O controle de memória pode ser foco de discussão, uma vez que é um recurso da máquina e portanto poderia estar classificado como entidade essencial. Como no caso da interface de E/S, assume-se a utilização de funções da entidade essencial, e o controle a este nível com um perfil mais lógico.

Abordagem semelhante pode ser dada ao controle de calendário (utilização de funções essenciais com controle lógico).

### 2.4.3 Entidade de serviço à aplicação

Situam-se neste nível as funções que são fornecidas diretamente à aplicação mas que podem existir ou não de acordo com as necessidades desta. Geralmente as funções desta entidade são compostas de serviços que controlam funções de outras entidades, ou que fazem uma interface mais adequada a elas.

A fim de fornecer uma máquina virtual à aplicação pode-se ter esta entidade como a "fronteira" entre a aplicação e a máquina. O modelo aqui descrito não coloca este requisito como imprescindível, entretanto pode ser utilizado caso se deseje uma visão totalmente encapsulada da divisão em "entidades". Neste caso, todas as funções passsíveis de acesso pelo usuário, mas classificadas nas entidades, deveriam ser utilizadas por interfaces existentes nesta entidade.

Esta entidade pode ser implementada como uma biblioteca de funções, fornecendo uma extensão da máquina hospedeira. Sua utilização pode se dar em duas visões: como parte da própria aplicação, ligada ao código executável, e na forma de uma complementação do software já existente na máquina.

### 2.4.4 Considerações sobre o modelo

As entidades apresentadas são autônomas no sentido em que encapsulam gerências que são vistas externamente através das interfaces às funções que executam. Não existe um caráter estritamente hierárquico uma vez que todas as funções de uma entidade podem ser usadas pelas demais e mesmo pela aplicação. As características da implementação de uma função são determinadas pelos serviços que a compõem, dentro da gerência que a fornece. A funcionalidade é mantida mesmo com a alteração da filosofia da execução de um serviço.

Nota-se que conceitos das várias abordagens apresentadas estão sendo

usados aqui. Por exemplo, o encapsulamento do modelo de objetos, a hierarquia do modelo em camadas e a programação macro/microscópica na diversidade da forma de implementação mantendo funcionalidade.

As relações de dependência interna e externa entre entidades serve como base para que se possa chegar a várias formas de núcleo. Também são importantes na agregação ou retirada de funcionalidades, buscando sempre reutilizar os elementos já presentes, a nível de serviços ou gerências.

O aspecto do contexto em que o núcleo será inserido é também considerado quando da definição das entidades. Para o contexto "tempo real" alguns compromissos devem ser assumidos, principalmente no escopo da entidade essencial. Por exemplo, busca de maior desempenho em atividades muito executadas, filosofias de preempção, prioridades, controle de tempo de execução de tarefas são ítens que devem ser levados em consideração.

No caso de operação tempo-real, outra característica para um núcleo é a composição software básico + aplicação como um produto final, possibilitando uma otimização nas partes críticas e uma operação dedicada para as partes específicas da aplicação.

Observa-se que a utilização de uma modelagem para construção de um núcleo tem como objetivo "organizar" o desenvolvimento deste tipo de software. O modelo proposto utiliza conceitos de diferentes abordagens, de forma a possibilitar que as questões específicas de *núcleos* sejam consideradas. Por exemplo, permitindo o encapsulamento de funções em entidades, o aspecto de implementação é isolado e pode ser trabalhado no sentido de otimização de código.

A entidade de serviço à aplicação permite a agregação de novas funções, acompanhando evoluções da máquina; a entidade básica procura ressaltar o aspecto de reusabilidade; e a entidade essencial concentra as dependências da máquina, separando-as do modelo lógico das demais funções fornecidas pelo núcleo.

O modelo, como um todo, permite a configuração do núcleo para dife-

rentes aplicações, o que será discutido no capítulo 4.

Nas figuras 2.2 - pag. 29 e 2.3 - pag. 30 encontram-se alguns modelos de formação de um núcleo a partir do modelo de entidades.

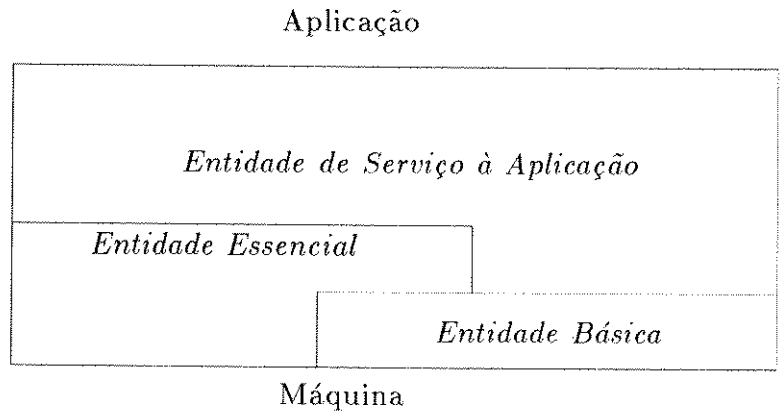



Figura 2.2: Relacionamento entre entidades do núcleo - modelo 1

æ

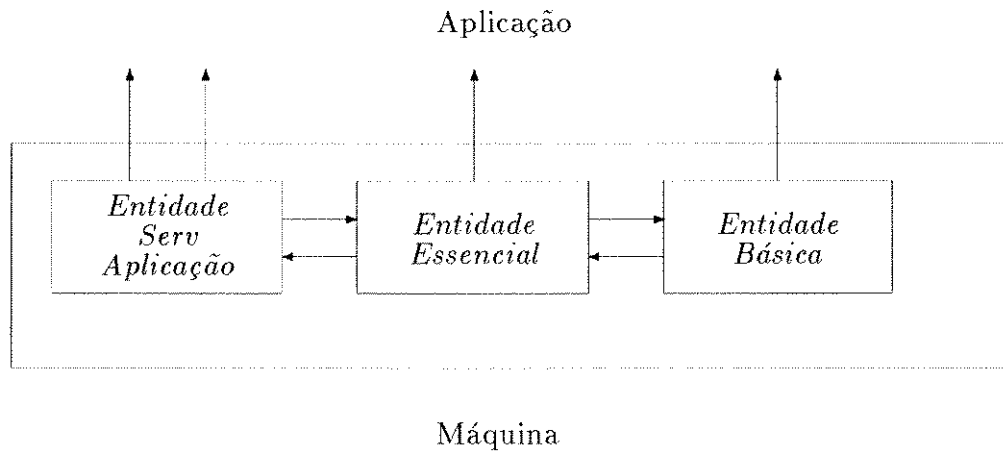



Figura 2.3: Relacionamento entre entidades do núcleo - modelo 2

æ

æ

# Capítulo 3

# Características e estratégias de implementação de entidades para construção de núcleos

Foi observado dentro do modelo de entidades proposto no capítulo anterior que:

- entidades são compostas de gerências;
- cada gerência fornece um conjunto de funções que caracterizam a entidade,
- gerências são formadas por serviços.

Neste capítulo serão apresentadas, por entidade, as características funcionais de diversas gerências e algumas estratégias de implementação.

No aspecto funcional será apresentada uma proposta sobre quais operações uma gerência deve englobar, tomando como orientação características de *núcleos*.

Como estratégia de implementação serão discutidas maneiras de fornecer as funções de uma gerência. Nesta discussão inclui-se também a escolha da linguagem a ser usada para a implementação das entidades. O uso de uma linguagem de alto nível é desejado principalmente para atender às necessidades de portabilidade. Dentro das linguagens disponíveis a mais indicada é a linguagem "C" [28], por suas características que permitem combinar as melhores funções de uma linguagem de alto nível com habilidades de linguagens de máquina.

## 3.1 Entidade Essencial

As gerências que compõem a entidade essencial são responsáveis pela interface física com a máquina hospedeira e pelo fornecimento das funções mínimas de um núcleo. Classificam-se como gerências desta entidade:

- a gerência de interface com o hardware
- a gerência de escalação de tarefas
- a gerência de sincronização e comunicação entre tarefas

### 3.1.1 Gerência de interface com o hardware

As funções nesta gerência são ligadas à programação do hardware da máquina e ao tratamento de interrupções, tanto ligadas a periféricos quanto ao controle de "tempo". Em geral, a máquina possui algum elemento hardware, como um contador, que é capaz de gerar a base para um controle de tempo.

A utilização de "tempo" em um núcleo permite a implementação de:

- agendas
- calendários
- "timers"
- controle de operações que levam à suspensão de tarefas, evitando casos de esperas infinitas.

Esta gerência é responsável também pela preparação para atendimento aos eventos gerados pelos recursos da máquina (controladores de tempo e de periféricos).

Uma outra função desta gerência é o controle do acesso a recursos comuns. Isto pode se dar a nível hardware, por exemplo via bloqueio e liberação de interrupções, ou software, através de elementos para coordenar acessos ou tarefas especiais de tratamento para cada recurso do sistema. [1].

[1] O controle por software ficaria a nível de entidade de serviço à aplicação

Esta gerência não oferece interfaces para todos os serviços qsue a compõem, uma vez que a programação hardware é preferivelmente encapsulada dentro da entidade, para evitar grande ligação aplicação/máquina.

Funções típicas desta gerência são:

- "inicia operação": executa os serviços de programação da máquina, e transforma o ambiente efetivamente em "multitarefa"
- "termina_operação": função inversa da descrita acima, retorna a máquina ao estado inicial através dos controles hardware apropriados. Termina a operação "multitarefa" do ambiente

Para o controle de acesso a recursos compartilhados, as funções possíveis seriam:

- "bloqueia_operação_multitarefa": garante à atividade (tarefa em execução) o domínio do processador, até a chamada de ...
- "libera_operação_multitarefa": operação inversa da descrita acima, retorna à operação multitarefa do núcleo

No tocante a acessos a entrada/saída de dados, esta gerência deveria prover os controles a nível hardware, como controle de uma interface serial ou de vídeo/teclado disponíveis na máquina hospedeira. Estas funções servem como insumos para operação da gerência de E/S da entidade básica. Os serviços ligados a estas ações são bastante dependentes da máquina hospedeira; a programação dos recursos da máquina deve ser suficiente para que a entidade básica possa operar somente sobre a lógica da função.

Quanto à memória, (que é um recurso da máquina), esta gerência deve prover o controle sobre um eventual controlador, ou um algorítmo de alocação/dealocação sobre a memória total. As funções deste serviço

vão ser usadas no escopo de entidade básica por um gerenciador de nível "lógico" (não dependente da máquina).

As funções mínimas, para memória, fornecidas neste nível seriam:

- "aloca": aloca um número requisitado de bytes
- "libera": libera uma quantidade anteriormente alocada de bytes
- "informa": informa memória disponível, em bytes

### Aspectos de implementação

Situam-se nesta gerência as ações mais dependentes da máquina. Vários serviços desta gerência são escritos na linguagem de montagem do processador, devido à necessidade de se programar diretamente o hardware da máquina alvo. Obtém-se desta forma maior facilidade de programação dos circuitos específicos da máquina, do contador "timer", e melhor performance em situações com restrições de tempo de execução, como trocas de contexto.

A este nível opta-se por uma operação via interrupções ou "polling", e iniciam-se as condições necessárias à operação multitarefa. A opção pelo uso de interrupções para a comunicação de um evento deve-se à rapidez de atendimento fornecida pelo hardware, se comparado a uma solução via "polling", cuja latência entre o acontecimento até o reconhecimento do evento pode ser alta.

Serão descritos a seguir aspectos de implementação referentes aos grandes grupos de serviços desta entidade.

1. Operações básicas

   Assumindo uma operação por interrupções, o seu atendimento é uma ação que vai ativar a gerência de escalação de tarefas,

caso se opte pela existência de preempção. A operação de troca de contexto de execução, por ocorrer imediatamente após o tratamento do evento, é parte desta gerência. A interface entre o serviço de atendimento e o de troca de contexto deve ser profundamente estudada, caso sejam escritas em linguagens diferentes (por exemplo, C e assembly).

Uma implementação efetua os seguintes serviços sobre a máquina:

- cria um intervalo padrão com o relógio da máquina, gerando uma base de tempo para a operação do núcleo. Esta base pode ser um intervalo tal que, a cada período, são verificadas as condições das tarefas presentes, e escolhida uma tarefa adequada para execução.
- bloqueio da operação do núcleo no tocante a trocas de contexto, garantindo a ação ininterrupta de uma tarefa. Isto implica em tratar interrupções normalmente, mas não se trocar de contexto de execução mesmo havendo condições para tal. A contagem do relógio continua enquanto há o bloqueio; na liberação da operação multitarefa, o controle de tempo global do núcleo é "corrigido" de forma a compensar o tempo em que não houve o tratamento completo da interrupção. A implementação do bloqueio da troca de contexto pode ser feita através de uma variável global, que tem um determinado valor que indica o bloqueio e outro que indica a liberação. Este valor é verificado antes da operação de chaveamento de contexto.
- libera operação multitarefa - libera trocas de contexto entre tarefas "prontas", utilizando o mesmo conceito de variável global.
- tratamento de interrupção, executando os serviços de:

* controle dos "tempos", caso seja uma interrupção de relógio; atualiza elementos dependentes do tempo
* controle da interface com um dispositivo, se é interrupção ligada a periférico
* verificação da condição das tarefas presentes no núcleo e a possibilidade de troca de contexto, caso haja preempção
* controle da troca de contexto, alterando o ambiente de execução. O tratamento referente à escolha de qual tarefa deve assumir o controle da máquina é feito por um serviço da gerência de escalação de tarefas. O serviço prepara o contexto para a tarefa a ser executada, recuperando pilha e registradores a ela relacionados
* controle da execução de atividades ligadas à interrupção antes do início da operação do núcleo.

2. Operações sobre memória

   Utilizando-se controle de memória via hardware ou através de facilidades da máquina, os serviços para programação e supervisão deste controle fazem parte desta gerência.

   Para controle de memória pode-se usar várias filosofias [29], [30]. No caso da máquina oferecer segmentação de memória, o controle pode ser efetuado através da alocação de determinados segmentos ao núcleo e outros à aplicação. Outra alternativa, independente de segmentação, é a manutenção de um mapa de alocação que, vendo a memória como um conjunto de blocos de tamanho fixo, indica quais os livres e cupados. Outro exemplo é um esquema controlado através de uma lista onde cada elemento aponta para blocos livres de diferentes tamanhos. A filosofia de alocação pode envolver técnicas de "garbage collection",

evitando fracionamento da memória em pequenas áreas. Alguns processadores já trazem embutidas gerências de memória, controlando privilégios para acesso ou fornecendo meios para se implementar paginação, ou então trabalhando com segmentação. A linguagem C, por exemplo, fornece um esquema de gerência de memória, cujo modo de operação é semelhante a manter-se uma lista ligada de informações sobre blocos de memória alocados pelo solicitante. Estas funções não têm qualquer proteção na alocação/dealocação dos blocos de memória. Seu funcionamento baseia-se na manutenção dos blocos livres em uma lista, juntamente a um cabeçalho que contém informações para a manutenção da lista e o tamanho do bloco que representa. A cada solicitação de memória um destes blocos é "entregue" na forma de um apontador para a área alocada, antes controlada pelo cabeçalho, que continua a fazer parte desta área, apesar de não acessada pelo solicitante. A liberação de memória recoloca o bloco na lista a partir dos dados da área do cabeçalho do bloco.

3. Operação sobre dispositivos de E/S

   O controle de dispositivos de E/S é encarregado, por exemplo, da conversão de um comando de E/S em seqüência de "escapes", ou da obtenção de caracteres via interface serial ou teclado. Podem ser fornecidos serviços como:

   - escrever um caracter no vídeo na posição do cursor
   - obter o caracter sobre o qual está o cursor
   - escrever uma cadeia de caracteres no vídeo, movimentando ou não o cursor
   - escrever um caracter sobre a posição do cursor
   - obter um caracter do teclado
   - obter posição do cursor

- coloca cursor em determinada posição no vídeo
- "rolamento" ("scroll") de uma região do vídeo

A forma de implementação destes serviços é fortemente ligada ao hardware disponível na máquina; no caso de uma máquina com poucos recursos, alguns deles seriam parte da entidade básica. Neste nível, situariam-se apenas os controles mais elementares do hardware.

### 3.1.2 Gerência de escalação de tarefas

Uma *tarefa* no núcleo é uma unidade de execução independente. É um conjunto de instruções que deve ser executado seqüencialmente, mas que contem operações disjuntas às demais ações do núcleo.

As tarefas existem, em um núcleo, a partir de sua criação pela aplicação, e podem estar em um de três estados:

- suspenso
- pronto
- executando

com variantes de acordo com o tipo de filosofia de execução de tarefas. O estado "suspenso" indica que alguma ação foi executada pela tarefa e que esta deve aguardar a ocorrência de um "evento" para que continuar sua execução. O estado "pronto" indica que a tarefa tem todos os requisitos necessários para sua execução. A tarefa que possui o processador está no estado "executando".

O controle sobre os estados de tarefas e a operação de distribuição do processador entre estas tarefas são feitos por um serviço desta gerência: o "escalonador". O escalonador é acionado quando existe

uma situação acusada por "tempo", por ocorrência de evento externo ou por solicitação do próprio usuário. Na operação do escalonador influem as necessidades da aplicação, principalmente se existirem requisitos de tempo real. Outros exemplos desta influência são a execução de algumas tarefas preferencialmente a outras e o controle de seu tempo de execução.

Esta gerência englobaria como funções:

- "cria_tarefa": prepara os controles para tratar o código definido como tarefa pela aplicação dentro dos princípios de execução multitarefa (código reentrante, execução independente);
- "destrói_tarefa": retira uma tarefa anteriormente criada do ambiente de execução e trata da correta liberação de quaisquer recursos associados a ela;
- "suspende_tarefa": coloca uma tarefa em estado "suspenso", por solicitação explícita da aplicação, e por um tempo por ela determinado, após o qual o próprio núcleo coloca-a novamente no estado "pronto";
- "resume_tarefa": recoloca uma tarefa anteriormente suspensa no estado "pronto", forçando sua reativação antes da ocorrência do evento determinado quando de sua suspensão;
- "reescalona": analisa as tarefas na estado "pronto" e a tarefa "executando" a fim de escolher para execução aquela com mais condições ou privilégios para execução;
- "muda_pri": altera a prioridade de uma tarefa.

As funções citadas acima partilhariam de serviços desta e eventualmente de outras entidades. Por exemplo, as funções de criação e terminação de tarefas poderiam utilizar as funções de gerência de

filas da entidade básica e os serviços de tratamento de ocorrências (interrupções) da gerência da interface com o hardware.

### Aspectos de implementação

A escalação de tarefas no núcleo é parte central e a que mais influencia a performance final do sistema. Diversos algorítmos podem ser utilizados e a determinação do melhor deles, de acordo com a aplicação, pode levar a um melhor resultado final.

Algumas decisões como prioridade, preempção, e número máximo de tarefas no núcleo são tomadas nesta gerência. Em geral, um dos conceitos mais importantes desta gerência é a existência ou não de preempção. O fato de se manter um esquema preemptivo implica em que uma interrupção do temporizador, por exemplo, pode mudar o ambiente de execução.

Faz parte do modelo de implementação a definição de "presença" de tarefas, que são compostas para o núcleo por um código (indicado pela aplicação) e uma estrutura de controle, que armazena o contexto da tarefa e informações para o núcleo. A este conjunto chama-se normalmente "descritor".

As tarefas são "escolhidas" para execução de várias formas, por exemplo:

- por prioridade
- por tempo de espera pelo processador
- pelo menor tempo de espera pelo processador
- pelo menor tempo de execução
- por partilhamento de posse do mesmo em tempos iguais entre tarefas

- por previsão de tempo de ocupação do processador por uma tarefa

ou combinações destas filosofias. A manutenção dos descritores no núcleo está ligada à filosofia de escalação de tarefas; podem ser ordenados em uma fila por prioridade, diversas filas, ou referenciados por outros elementos de controle do escalonamento, dependendo do tipo de implementação. Para o tratamento dos descritores em filas podem ser utilizados tratamentos genéricos, através de funções da entidade básica (como descrito adiante).

Apresentam-se abaixo algumas das filosofias existentes para escalonadores [31]:

- *Preemptivo, uma fila de descritores de processos*
  Neste algorítmo há uma fila de descritores de tarefas ordenada através de uma "chave" escolhida. Esta "chave" espelha o esquema de prioridades a ser utilizado, que pode ser o instante de criação da tarefa, o tamanho da mesma, menor tempo de execução, ou um valor dado pela aplicação, entre outros. A prioridade é função de parâmetros definidos na criação da tarefa. Tarefas com prioridades iguais são escolhidas para execução mediante um esquema onde a tarefa com maior tempo de espera pelo processador é executada antes. É um esquema adequado para ambientes com uma grande faixa de valores de prioridades.
- *Preemptivo, várias filas de descritores de processos*
  Neste caso tem-se a multiplicação de várias filas, tantas quantas forem as prioridades existentes no núcleo. A ativação de tarefas se dá varrendo as filas em ordem de prioridade descendente. O primeiro elemento encontrado representa a tarefa que é colocada em execução. A colocação de um descritor de tarefa em uma

fila é feita em um esquema FIFO. Um escalonador deste tipo é o "Feedback queues": uma tarefa é colocada em uma fila de menor prioridade a cada vez em que é executada. Este tipo é adequado a tarefas com tempos pequenos de processamento.

- *FCFS - First come first served*

  É um algorítmo não preemptivo. A um pedido de ativação, se o processador estiver livre, a tarefa solicitante toma-o completamente até o fim de sua execução. Pedidos posteriores são colocados à espera, em uma fila FIFO. Este algorítmo, apesar de sua simplicidade, tem uma performance baixa caso o tempo de execução das tarefas seja desbalanceado (tempos de execução diferentes entre si). Caso as tarefas sejam curtas e "bem comportadas"[2], pode-se ter um desempenho razoável [3], [1].

- *Time-sharing*

  Neste tipo de escalonamento cada tarefa presente tem para sua execução fatias de tempo pré-definidas. Ao fim de cada "fatia" a tarefa é necessariamente suspensa. Sistemas com esta orientação são adequados a aplicações sem requisitos de tempo real. Existem variações deste algorítmo, como a alocação de "fatias" de tempo diferenciadas para cada tarefa.

- *Menor tempo de execução*

  Neste caso a seleção de uma tarefa para posse do processador é feita baseada em alguma informação, dada pela própria, do tempo em que vai ocupar a máquina. Algumas restrições de tempo são especificadas pela tarefa, como [31]:

  * tempo de chegada, que é o instante em que a tarefa foi criada

[2]liberando o processador a tempos médios relativamente constantes

* tempo de pronto, que é o instante no qual a tarefa pode iniciar sua execução
* o tempo de sua execução
* o tempo máximo para o encerramento da operação da tarefa

Este esquema é interessante nos casos onde as tarefas devem ter seu tempo de execução bem determinado, adequando-se a casos de aplicacões em tempo real com fortes restrições de tempo.

### 3.1.3 Gerência de sincronização e comunicação entre tarefas

Esta gerência deve prover uma via mínima através da qual as tarefas da aplicação possam se comunicar e/ou sincronizar. A sincronização, neste contexto, é vista como uma forma particular de comunicação, onde uma "mensagem" não contém dados mas representa uma condição de sincronismo entre tarefas.

Poderiam existir várias formas de ação para esta gerência; porém, neste modelamento, formas de mais alto nível seriam implementadas na entidade de serviço à aplicação. Sob este ponto de vista, esta gerência, na entidade essencial, proveria uma forma genérica de comunicação.

São formas comuns para comunicação e sincronização:

- o uso de "sinais" que "trafegam" entre tarefas
- o uso de "portas" ou "caixas postais"
- o uso de áreas comuns de memória

e as funções que, em qualquer forma de comunicação, devem ser providas são:

- criação de um meio de comunicação
- terminação de um meio de comunicação
- envio de uma mensagem para uma tarefa utilizando o meio de comunicação
- recebimento de uma mensagem por uma tarefa através desse meio de comunicação

A utilização de "sinais" para a comunicação implica em definir o formato de um sinal, o que pode particularizar o serviço de forma não condizente com a aplicação. Uma característica de utilização de "sinais" é o tempo gasto no seu tratamento, tornando a comunicação bastante "abstrata" do ponto de vista da máquina.

O uso de comunicação via memória partilhada é o mais ligado à arquitetura da máquina. É bastante frágil do ponto de vista de proteção; um erro de endereçamento no acesso a estas áreas pode ser totalmente prejudicial.

A comunicação via "portas" alia um nível abstrato suficiente para prover proteção e não tão particular que não possa ser adaptado às necessidades da aplicação. Nesta forma, assume-se que uma "porta" é uma área de armazenamento controlado de dados, acessada por qualquer tarefa de forma organizada.

Seriam fornecidas nesta entidade funções genéricas, de forma a dar insumos para a construção de outros serviços lógicos, tanto a nível da entidade de serviços à aplicação como pela própria aplicação. Assim, a escolha da implementação de portas nesta gerência é adequada.

### Aspectos de implementação

Uma *porta* é considerada como um meio controlado de armazenamento de dados, que podem ser de qualquer formato. Para fins de implementação isto pode materializar-se em filas controladoras de elementos, de apontadores para elementos, tabelas associando o elemento às tarefas enviadora e receptora, ou como áreas comuns de memória controladas pelo próprio núcleo. A diversidade de filosofias mostra que deve ser tomado como parâmetro de escolha do algorítmo de implementação aquele mais apropriado a obter melhor desempenho na operação, ou que puder ser elaborado mais facilmente.

A existência de uma gerência de filas no contexto da entidade básica sugere a utilização de algorítmos baseados em filas. A forma mais genérica de implementação é a associação de uma "porta" a uma fila, utilizando a gerência de filas para os controles de criação, deleção, inserção e retirada. Cada elemento da fila, nesta implementação, é um apontador para o objeto colocado na porta. Este pode ser uma mensagem, uma linha de caracteres, ou uma estrutura qualquer.

Uma característica que pode ser incorporada a esta gerência é a associação de tempo à fila; neste caso, é controlado o tempo de espera por mensagem de uma tarefa em uma porta. O serviço de atendimento a interrupções também agiria sobre as "portas", atualizando um fator de "tempo" a elas associado. Ao esgotamento do tempo de espera da(s) tarefa(s), esta(s) é(são) colocada(s) automaticamente no estado "pronto", evitando esperas infinitas.

As características genéricas deste tipo de implementação permitem que outros serviços possam utilizá-lo para fornecer novas funções. Isto pode se dar tanto a nível de núcleo, com uso pelas demais entidades, quanto a nível de aplicação, mesmo sem tratamento explícito por

alguma entidade.

### 3.1.4 Considerações sobre as gerências da entidade essencial

As gerências apresentadas não são únicas nem conclusivas, mas procuram listar requisitos mínimos para construção de um núcleo. Observando somente esta entidade pode-se concluir que já é fornecida uma máquina virtual mínima para a aplicação, com características multi-execução, apenas com as funções previstas.

Nas tabelas 3.1 - pag. 49, 3.2 - pag. 50 e 3.3 - pag. 51 estão relatadas de forma resumida as funções desta entidade, de acordo com a gerência onde foram definidas.

| função | atividade | interface externa |
|---|---|---|
| controle do hardware | programação da máquina | não |
| controle memória | ação sobre tempos e solicitações de recursos e E/S | sim |
| inicia_operação | inicia controle hardware, prepara operação multitarefa | sim |
| termina operação | termina operação multitarefa, volta máquina ao normal | sim |
| bloqueia_op_ multitarefa | garante execução initerrupta de uma tarefa | sim |
| libera_op multitarefa | libera operação multitarefa após bloqueio | sim |

Tabela 3.1: Gerência de interface com o hardware

| função | atividade | interface externa |
|---|---|---|
| cria_tarefa | inicia controles para execução de ação da aplicação como "tarefa" | sim |
| termina tarefa | retira a ação da operação multitarefa | sim |
| suspende_tarefa | coloca tarefa em estado suspenso por tempo determinado | sim |
| resume_tarefa | força colocação de tarefa suspensa no estado pronto | sim |
| reescalona | analisa tarefas prontas, escolhe mais conveniente para execução | sim |
| muda_pri | altera privilégio de execução de uma tarefa | sim |

Tabela 3.2: Gerência de escalação de tarefas

| função | atividade | interface externa |
|---|---|---|
| cria_porta | inicia área de controle de comunicação entre tarefas | sim |
| deleta_porta | dealoca área de controle de comunicação | sim |
| envia_mensagem | coloca dados na área de comunicação | sim |
| recebe_mensagem | lê dados na área de comunicação | sim |

Tabela 3.3: Gerência de sincronização e comunicação

## 3.2 Entidade básica

As gerências que compõem a entidade básica têm como característica comum o fato de poderem ser parte integrante de outras gerências nas demais entidades, por fornecer serviços suficientemente genéricos para utilização em vários contextos. Outra característica é a diversidade de formas com que uma gerência desta entidade pode ser implementada, o que permite a exploração de vários algorítmos para uma mesma funcionalidade.

Alguns dos serviços desempenhados por esta entidade são agrupados em:

- gerência de filas
- gerência de entrada e saída
- gerência de memória
- gerência de calendário

Entretanto, outras gerências podem ser incluídas, caso tenham características genéricas.

### 3.2.1 Gerência de filas

Existem várias formas de controle de filas, cada qual voltada para um tipo de necessidade. Algumas funções sempre são fornecidas, independente da filosofia de implementação do controle das filas. Partindo de um conceito de criação de um elemento do tipo FILA, as seguintes funções seriam fornecidas por esta gerência:

- "cria_fila": cria e inicia o elemento FILA, a partir de algumas informações como

    * número máximo de elementos da fila
    * tamanho de cada elemento da fila
    * filosofia de ordenação dos elementos da fila
- "deleta fila": libera a FILA, caso não haja elementos nela armazenados;
- "insere_fila": insere um elemento em FILA
- "retira_fila": retira um elemento da FILA
- "remove_fila": retira um elemento específico de FILA
- "busca_fila": procura um elemento específico em FILA
- "num_fila": informa o número de elementos em FILA

### Aspectos de implementação

Algumas das formas mais usuais para trabalho sobre filas são os esquemas onde o instante de colocação do elemento na fila influencia na ordenação da mesma. A operação usual é retirar o primeiro elemento inserido ("FIFO - First In First Out"), ou o último ("LIFO - Last In First Out"). Outro esquema é a ordenação da fila por um algorítmo do tipo "heap" [32], ou fila por prioridade. Neste caso, é utilizado para ordenação da fila uma "chave" definida no elemento - e não o instante de colocação na fila. Embora ineficiente, é possível, com este esquema, manter uma fila FIFO ou LIFO; ou seja, este algorítmo de implementação é bastante genérico, pois permite que diversas informações possam servir para ordenação da fila.

Uma fila ordenada por prioridade pode ser representada como uma árvore binária com as seguintes propriedades:

1. todos os elementos "filhos" têm um valor menor que os pais

2. a árvore é tão balanceada quanto possível (a diferença no "peso" das folhas é no máximo 1)
3. as "folhas" são inseridas na árvore da esquerda para a direita até completar uma linha, após o que outra linha é iniciada.

Uma característica deste tipo de fila é a necessidade de reordenação a cada inserção (um processo chamado "reheaping"), a fim de manter a característica *1*. Os ítens *2* e *3* destacam a conveniência de representar a árvore como um vetor onde o elemento no nó $N$ tem "filhos" nos nós $2N + 1$ e $2N + 2$.

O problema principal desta implementação é a necessidade prévia de conhecer a quantidade de elementos que a fila vai conter; outros problemas são a retirada de um nó "interno" (ou seja, uma deleção aleatória) e a reunião de duas filas ("merge"). Como vantagens, obtém-se ganho em eficiência onde são necessárias apenas funções de inserção e retirada de elementos; não há busca pela posição de inserção, nem pelo elemento a ser retirado, pois estão sempre em posições fixas. Apesar das cópias de elementos durante o processo de reorganização ("reheap"), estas não ultrapassam o valor $log_2 N$. Para maior otimização do tempo das cópias armazenam-se, normalmente, apontadores para elementos.

### 3.2.2 Gerência de entrada e saída

Um núcleo não oferece, a princípio, interfaces para memória de massa; ou seja, não é definido um sistema de arquivos nem acesso a discos ou outros elementos de armazenamento de dados. As funções mínimas de E/S fornecidas são ligadas à visualização e à entrada de dados, normalmente por uma via serial, e são utilizadas para depuração e informação sobre o sistema. São usadas funções providas pela entidade

essencial para interface aos elementos de E/S existentes na máquina. Esta gerência caracteriza-se pelo tratamento lógico dos dados, o que a situa no escopo de entidade básica. O tratamento dos dados pode ser desde a tradução de um número em seu ASCII correspondente até o controle de "janelas", com cores ou outros recursos, no dispositivo de saída disponível.

Algumas funções que podem ser fornecidas por esta gerência são:

- "lê_car": lê um caracter do dispositivo de entrada de dados (usando a interface hardware controlada na entidade essencial)
- "inthex": transforma um dado lido em seu ASCII correspondente
- "lê_lin": lê um conjunto de caracteres do dispositivo de entrada até um determinado caracter ou um número máximo, permitindo edição destes dados ("backspace" e "del")
- "inicia_vídeo": inicia vídeo ligado à máquina
- "cria_janela": cria uma "janela" no vídeo
- "limpa_janela": limpa a "janela"
- "escreve_janela": escreve em janela

e outras funções de acordo com necessidades da aplicação e disponibilidade de recursos da máquina.

### Aspectos de implementação

A busca de uma padronização de interfaces de E/S é uma constante no desenvolvimento de sistemas que devem operar sob ambientes distintos, sobretudo se houver grande necessidade de interação com o

usuário. Uma das opções disponíveis para esta padronização são pacotes de rotinas que gerenciam "janelas". Estes pacotes são compostos, em geral, por um conjunto de funções que fazem a interface com o hardware (classificáveis como "entidade essencial") e um conjunto de operações lógicas (situadas como "entidade básica). Estas operações permitem ações como a movimentação do cursor na tela, criar e deletar janelas, escrever textos em áreas de janela, entre outras. As janelas tratadas pelas rotinas podem se superpor, e para cada janela é controlado "scroll", "wraparound", cor, visibilidade, atributos de caracteres, entre outros fatores. A filosofia de implementação normalmente visa otimizar o acesso à saída, trabalhando com o envio do número mínimo de caracteres a cada alteração, o que é obtido através da manutenção interna de duas imagens da tela: a que é a imagem real, e uma que serve como "rascunho", onde são efetuadas as mudanças. A um pedido de atualização o conteúdo das duas imagens é comparado, e só o que difere nesta comparação é enviado para o dispositivo de saída. Pode ser utilizado para implementação desta gerência um pacote deste tipo, oferecendo, por exemplo, as seguintes funções:

- iniciação do pacote
- terminação do pacote
- entrada de caracteres não bufferizada
- cria nova janela
- salva conteúdo de janela existente
- abandona janela criada
- deleta janela
- esconde janela

- mostra janela
- move janela para posição absoluta
- move janela p/ posição relativa
- acionamento do scroll na janela especificada
- idem a scroll p/ wrap-around
- faz refresh da tela
- faz refresh da janela
- move cursor para posição especificada
- move janela para posição especificada
- obtém caracter e ecoa para saída de dados padrão
- ídem, ecoa na janela especificada
- retorna o caracter na posição do cursor
- escrita formatada em janela
- coloca caracter em janela
- limpa janela
- limpa tela
- scroll da janela, uma ou n linhas

### 3.2.3 Gerência de memória

A gerência de memória nesta entidade controla a alocação e dealocação de frações de memória, tanto fornecidas à aplicação como ao próprio núcleo durante a execução de suas funções.

Esta gerência forneceria como funções, utilizando serviços da entidade essencial:

- "pede mem": aloca memória para um processo ou para o núcleo. A memória, neste caso, é entregue como um bloco indivisível, de tamanho possivelmente maior (em bytes) do solicitado, caso seja usado um esquema de alocação de quantidades mínimas pela entidade essencial, para fins de eficiência
- "libera_mem": libera uma quantidade de memória anteriormente alocada por "pede_mem"
- "mem_disp": informa a quantidade de memória livre disponível no momento
- "mem_usada": informa a quantidade de memória usada até o momento

O caráter *básico* desta gerência é dado pela possibilidade de se poder usar diretamente as funções providas pela entidade essencial, mas se efetuar um controle maior sobre elas, ou também na possibilidade de fornecer outras funções como controle de memória do núcleo ou da aplicação, controle de direito de acessos, interface para eventual controle de paginação e interface com memória de massa, entre outros.

### Aspectos de implementação

Assumindo uma implementação a nível de entidade essencial utilizando as rotinas da biblioteca da linguagem a ser usada para implementação ("C"), uma gerência nesta entidade pode prover os controles não efetuados ainda sobre a memória. Um exemplo é o da verificação da validade dos blocos dealocados, o que não é feito a nível de biblioteca e pode levar a erros graves em casos de liberação de áreas não previamente alocadas. Este controle é relativamente fácil, bastando manter uma tabela de apontadores para as áreas de memória

alocadas (onde o apontador é a resposta da função "pede mem"), e buscar nesta tabela os apontadores referentes às áreas a serem dealocadas. Caso este valor não exista na tabela, acusa-se o erro. Um outro controle que pode ser colocado nesta tabela é a identificação do solicitante da memória (núcleo ou aplicação), de modo a verificar e permitir o uso de quantidades diferenciadas de memória.

### 3.2.4 Gerência de calendário

Utilizando o controle de tempo fornecido pela gerência de interface com hardware é possível a implementação de um calendário. Através deste calendário obtém-se funções de agenda e de informação de tempo real.

Funções desta gerência poderiam ser:

- "obtém_hora": informa a hora atual
- "seta hora": entra hora atual
- "cria_timer": cria um elemento capaz de fazer uma temporização em tempo real
- "liga_timer": inicia a temporização em um "timer" criado
- "desliga timer": interrompe a temporização em um "timer"
- "deleta_timer": deleta um "timer"

Estes serviços poderiam ser associados a elementos do núcleo para controle de eventos, além de utilização pela aplicação.

#### Aspectos de implementação

Uma forma de implementação desta gerência é através da definição de estruturas com controles para tempo e recursos a elas conectados.

A cada interrupção referente ao relógio, o serviço de tratamento da entidade essencial deveria atualizar estas estruturas.

Outra forma de prover estes serviços é utilizando a gerência de sincronização e comunicação, se contiver o controle de tempos em portas. Uma porta seria tratada como "especial", não recebendo mensagens; o mecanismo de tempo associado à espera de uma mensagem nesta porta, gerando "timeouts", serviria como uma marcação de intervalos adequados à implementação do calendário. A tarefa "suspensa" nesta porta especial conteria os controles para o fornecimento de data/hora reais; outras tarefas seriam associadas a "timers". Para a implementação de "timers" sua criação equivaleria a criar uma tarefa "timer"; a ligação, em colocar a tarefa criada à espera em uma porta especial, com tempo associado adequado; e o desligamento, a deleção desta tarefa. Para hora real, haveria uma tarefa sempre presente no núcleo, que receberia parâmetros para inicialização e forneceria informações através de uma outra porta, convenientemente controlada por serviços da gerência.

### 3.2.5 Considerações sobre a entidade básica

As gerências desta entidade procuram fornecer serviços que possam ser usados em vários contextos, ou seja, funções básicas.

Busca-se nesta entidade obter um fator de reusabilidade de funções do núcleo, separando-as em ações genéricas.

Assim como para a entidade essencial, esta discussão pretende formar um perfil das funções a serem fornecidas, não sendo portanto uma visão "fechada" e única.

Na tabela 3.1 - pag. 61 são apresentadas resumidamente as funções das gerências aqui descritas.

| gerência | função |
|---|---|
| gerência de filas | cria_fila |
| | deleta_fila |
| | insere fila |
| | retira_fila |
| | troca_fila |
| | remove_fila |
| | busca fila |
| | vê_fila |
| | num_fila |
| gerência de E/S | lê_car |
| | lê_lin |
| | inthex |
| | inicia_vídeo |
| | cria_janela |
| | limpa_janela |
| | escreve_janela |
| gerência de memória | pede_mem |
| | libera mem |
| | mem disp |
| gerência de calendário | obtém hora |
| | seta_hora |
| | cria_timer |
| | liga_timer |
| | desliga timer |

Tabela 3.4: Gerências da entidade básica

## 3.3 Entidade de serviço à aplicação

Nesta entidade situam-se funções que, utilizando as demais entidades, fornecem serviços adequados à aplicação. A necessidade de identificação entre a aplicação e estes serviços surge quando se busca otimização da execução de funções, ou então um nível de abstração que "esconda" as demais entidades do usuário final. Para que se leve o fator reusabilidade em consideração, esta entidade caracteriza-se por fornecer interfaces para funções de outras entidades ou por usá-las.

Algumas das gerências que podem existir nesta entidade serão descritas a seguir, embora para cada tipo de aplicação possa haver uma característica distinta que leve à necessidade de outras gerências.

### 3.3.1 Gerência de semáforos

O semáforo é um mecanismo para sincronização de eventos e controle de acesso a recursos que operam em exclusão mútua. Uma atividade que envolva a consulta a um semáforo pode levar à suspensão da tarefa que a efetuou, até que haja condição para sua liberação. Esta condição é comunicada ao controle do semáforo, que então passa o estado da tarefa para "pronto". A suspensão de uma tarefa por espera em um semáforo pode ter supervisão de tempo, para evitar suspensões por esperas infinitas. Conceitualmente são definidos dois tipos de semáforos: o binário e o contador. O primeiro permite que apenas uma tarefa tenha condições de execução quando da sua consulta, enquanto o contador permite um número variável de tarefas "prontas" após sua consulta.

As funções desta gerência seriam:

- "cria_semáforo": cria e inicia a área de controle de semáforo

- "sinaliza_semáforo": indica disponibilidade para passagem pelo semáforo. Se existem tarefas suspensas, uma delas vai ser colocada no estado "pronto"
- "suspende_semáforo": consulta o semáforo, e se ele não "permite" o prosseguimento da execução da tarefa (não há condição de disponibilidade) a mesma é suspensa
- "deleta_semáforo": "termina" a área de controle de um semáforo

### 3.3.2 Gerência de informações

Esta gerência fornece informações sobre a operação do núcleo, como número de tarefas ativas e suspensas, número de semáforos, número de portas ocupadas, entre outros. Serve como insumo para atividade de ajuste do núcleo à aplicação.

### 3.3.3 Gerência de erros

Esta gerência pode tratar erros ocorridos nas entidades básica e essencial de forma a afetar minimamente a aplicação. Com isso procura-se manter uma uniformidade na execução das funções do núcleo mesmo sob condições anormais. Algumas das atitudes tomadas nesta gerência são:

- a identificação da falha e de seu causador
- a ação sobre elementos dependentes da falha para recuperar a condição estável anterior
- informar convenientemente a aplicação

### 3.3.4 Gerência de canais de comunicação

Pode ser requisito da aplicação a existência de um meio lógico unidirecional para comunicação entre tarefas. Um "canal" pode ser elaborado para este fim, usandos os serviços da gerência de comunicação entre tarefas da entidade essencial e fazendo-se um controle lógico sobre eles.

### 3.3.5 Aspectos de implementação da entidade

As gerências desta entidade são essencialmente interfaces às funções das demais entidades, ou implementações de serviços particulares à uma aplicação.

No caso da gerência de semáforos a implementação pode ser feita no mesmo esquema da discutida para a gerência de calendário. Ou seja, utilizando portas, mas com a característica do tamanho da porta (quantidade de tarefas que podem estar suspensas nela) funcionando como direito de acesso a um recurso controlado pelo semáforo. A gerência de informações pode usar as diversas funções como "tarefa ativa", "mem disp", "mem usada", "obtem hora", e controlar uma saída conveniente através das funções da gerência de E/S. Concepção análoga pode ser usada pela gerência de erros.

### 3.3.6 Considerações sobre a entidade de serviço à aplicação

Esta entidade tem como função básica servir como interface às funções das demais entidades, de modo a torná-las mais facilmente utilizáveis pela aplicação. Além disso, um aspecto de portabilidade existe se for

considerado que a aplicação pode ver seu ambiente de execução como a esta entidade, sem "contato" com a máquina física. Esta entidade é bastante maleável, sujeita a inclusões e retiradas de funções. Eventualmente pode haver um esquema de emulação de gerências, visando o fornecimento de um ambiente mais genérico.

# Capítulo 4

# Configurabilidade de núcleos no modelo de entidades

O modelo desenvolvido nos capítulos anteriores tem como objetivo principal dividir as partes que compõem o núcleo de forma clara e independente. Esta divisão é feita com base no tipo de funcionalidade que o núcleo tem para cada ação por ele desempenhada - por exemplo, o controle dos elementos hardware, o fornecimento do ambiente multitarefa, a interface com entrada e saída. Estas partes podem ser separadas de modo a serem independentes do ponto de vista de implementação. Tomando cada parte como um bloco estanque, observa-se que é viável a existência de diversas filosofias de implementação para estes blocos, mantendo interfaces para os demais componentes do núcleo. O acréscimo de novas funções representa, neste esquema, a associação de mais blocos ao conjunto já existente. O uso de funções já fornecidas por outros blocos ou de algorítmos de domínio público é desejado, reforçando a idéia de reusabilidade [33], de procurar não "reinventar a roda", atacando problemas com soluções já conhecidas. Exceto para a interface direta com o hardware, estes blocos são independentes do ambiente em que se encontram, por implementarem funções lógicas, estarem escritos em uma linguagem de alto nível, e terem interfaces bem definidas entre si. Buscou-se obter com o modelo desenvolvido as seguintes características:

- a reusabilidade
- a expansibilidade
- a portabilidade

Além destas características, procurou-se possibilitar a configuração de diferentes núcleos, a partir de uma mesma base de código. Para isso, exige-se independência entre as funções que o compõem, conhecimento preciso das interfaces fornecidas por elas, e um controle a nível geral para a composição destes módulos. Um conjunto de código

independente de máquina, flexível, otimizado e um controle para o fornecimento de diferentes agrupamentos de funções proporciona uma ferramenta poderosa para diversos tipos de aplicações. É deste tópico que tratará este capítulo.

## 4.1 Porque um núcleo configurável ?

A necessidade de um núcleo e suas características já foram abordadas no capítulo 1. Lá chegou-se à elaboração de um modelo para construir um núcleo reunindo aspectos interessantes de diversas abordagens. Este esforço teve como objetivo principal permitir que, isolando-se convenientemente as partes do modelo, fosse possível agrupá-las de formas diferentes ou mesmo implementá-las de forma alternativa. Ou seja, *a proposta deste trabalho é que existam várias estratégias para a implementação dos elementos componentes de um núcleo, suficientemente definidas e isoladas a ponto de serem compostas de acordo com as necessidades de determinada aplicação.*

O porquê deste objetivo justifica-se pela diversidade de aplicações que necessitam de um ambiente multitarefa mas que, em geral, estão apoiadas sobre hardwares distintos. A tendência geral de controle na área de automação é o uso de sistemas baseados em microprocessadores suportados por núcleos com características de tempo-real [34], [27], [35], [36]. Entretanto, um núcleo neste enfoque deve ter o número estritamente necessário de funções para evitar problemas devido a restrições de tempo, e deve ser adequado à quantidade de memória disponível para seu funcionamento. Para cada aplicação existem ainda alguns detalhes que particularizam o código do núcleo a um projeto. As alternativas comuns são a compra de um núcleo ou a adaptação de um já existente. No caso de compra de software existem alguns incoveni-

entes, como a necessidade de se aprender uma nova interface e de se ter muito mais funcionalidade do que a realmente necessária, o que pode levar a tamanho de código ou performance inadequados. O aspecto de versatilidade também é comprometido - um núcleo particular é "proprietário" de seu código e de suas interfaces, podendo ser mais facilmente trabalhado para adaptação ou transporte. A adaptação de códigos prontos é muito dependente de como o mesmo foi estruturado. A experiência mostra que poucas partes são realmente reutilizadas caso haja forte interação entre elas. Em geral a nova aplicação exige exatamente o que não foi previsto no projeto original do núcleo - suporte a um novo processador, outras funções lógicas, outras interfaces externas. A preocupação com portabilidade do núcleo entre ambientes pode ser vista em [37], [38], [39], [40].

Outro objetivo deste trabalho é definir uma interface núcleo/aplicação genérica. Isto é viável a partir dos conceitos de entidade de serviços à aplicação que, essencialmente, fornecem um ambiente virtual independente de implementação à aplicação.

É interessante observar que existe uma tendência para o desenvolmento próprio de software básico para sistemas de automação e controle [41], que se fundamenta nas questões de simplicidade, versatilidade e facilidade de integração com a aplicação, se comparado ao uso de sistemas prontos. Nesta visão o uso de um núcleo configurável é bastante interessante, pois acrescenta a estas características a reusabilidade de soluções, um conjunto flexível de funções e a possibilidade de se adequar mais facilmente às necessidades da aplicação.

## 4.2 O princípio da configuração

A configuração proposta para o núcleo se baseia em duas filosofias, à semelhança da filosofia "programming-in-the-large" e "programming-in-the-small".

Uma primeira filosofia é a "microscópica", onde a configuração envolve a alteração da forma de implementação das gerências e serviços. Como exemplo, a alteração do algorítmo de escalonamento ou do tratamento de filas, mantendo a mesma interface funcional; muda nesta visão o mecanismo de construção dos módulos que fornecem serviços. A relativamente ampla teoria de sistemas operacionais e os dados obtidos a partir de estimativas tiradas de vários núcleos, gerados a partir de algorítmos distintos, fornecem os insumos para que se ajustem as funções da melhor maneira frente à aplicação. As alterações a nível "microscópico" são, a princípio, necessárias para obter melhor desempenho ou reduzir o tamanho do código do núcleo.

A filosofia "macroscópica" configura o núcleo a nível de funções, mantendo o conceito de entidade mas acrescentando ou retirando funções das mesmas. Esta manipulação se dá a nível da presença ou não de algumas gerências nas entidades. Alguns elementos estão sempre presentes (os que compõem a entidade essencial mínima), enquanto outros são sujeitos às necessidades da aplicação. A eventual emulação de funções pode ser implementada, de forma a responder ao solicitante adequadamente a não existência daquele serviço, ou através de informações de quais as funções presentes.

## 4.3 Propostas para um sistema configurador de núcleos

A partir do modelo que supõe independência entre as partes componentes do núcleo, um sistema configurador é definido como o controlador destes módulos para a composição de um núcleo. Um configurador deve:

- ter o conhecimento das relações entre os módulos do núcleo. Esta inter-relação existe no modelo de "entidades" definido, uma vez que há a reutilização de serviços entre funções e a necessidade de emulação de funções eventualmente não presentes.
- interagir com o usuário, obtendo as diretrizes da aplicação para a geração do núcleo
- controlar a existência de serviços no núcleo, tanto quanto a funções imprescindíveis como evitando a duplicação de serviços semelhantes.
- gerar o núcleo final, levando em consideração os ítens acima, de forma automática, havendo interação do operador apenas em uma fase preliminar.

Do ponto de vista do usuário as opções para ação sobre o núcleo são a escolha de funções presentes/não presentes e de "quantidades" de elementos. O configurador pode auxiliar informando ao usuário resultados de configurações anteriores quanto a tempos de execução, tamanho de código ou utilização de serviços, ou ainda propondo configurações, dadas as características da aplicação. O requisito mínimo de configurador é produzir o código final com características de núcleo [42]. Das várias estratégias possíveis para sua implementação, serão

propostas duas neste trabalho: uma em que se utiliza uma forma tradicional baseada em tabelas, e outra onde se lança mão de um sistema especialista.

### 4.3.1 Implementação de configurador baseada em tabelas

Este processo de configuração do núcleo se caracteriza por envolver manipulação de textos e gerenciar uma estrutura lógica para representar as interdependências internas do núcleo. A base de dados utilizada pelo configurador é uma tabela, na forma de um texto, onde se colocam as restrições e a inter-relação entre elementos do núcleo. Esta base de dados é interpretada de forma a compor uma estrutura hierárquica que é "navegada" durante o processo de configuração. Os dados que exprimem inter-relação são obtidos de arquivos no início da operação.

A utilização do configurador se inicia a partir de um diálogo com o usuário, através do qual são apresentadas as opções dentro de uma hierarquia que representa o modelo de construção do núcleo. É apresentada a opção de seleção de entidade a ser configurada, depois gerências e valores de constantes a ela associadas, dentro de uma visão "macroscópica" seguida de uma "microscópica". São informados também quais e como são as características dos elementos que devem estar sempre presentes e dos que podem ser incluídos opcionalmente. No tocante ao valor das constantes é apresentado um valor "default", e testada a opção do usuário contra um limite possível para o mesmo, limite este definido na base de dados.

O configurador gera, ao fim da sessão, um arquivo onde encontram-se os nomes dos módulos que serão utilizados na geração do núcleo, e

controla automaticamente o processo de compilação, ligação e produção do núcleo.

Um algorítmo possível para a implementação do configurador neste esquema é apresentado na figura abaixo:

```
PROGRAM

  apresenta opcoes de entidades do nucleo
  obtem selecao de entidade ou geracao de nucleo

  FACA ENQUANTO  selecao de entidade

    CASE entidade OF

    (essencial):
          apresenta opcoes de gerencias
          obtem selecao de gerencia

          CASE gerencia OF

          (...):
          DO
          apresenta  opcoes de constantes a configurar
                  obtem selecao de constante
                  obtem valor da constante

                  SE valor invalido
                        informa erro
                        EXIT
```

```
                FIMSE

                apresenta servicos associados e suas opcoes
                obtem opcoes
                seleciona arquivos relacionados
                          para integracao ao nucleo
        OD

  (basica):

        /* procedimento analogo */

  ENDCASE

  (servico a apliacacao):

                /* procedimento analogo */

  ENDCASE

ENDFACA

prepara arquivo de condicoes para compilacao
controla a compilacao de modulos
controla ligacao dos modulos
controla a producao da biblioteca
informa as caracteristicas do nucleo
gera arquivo de informacao de caracteristicas
termina sessao
```

```
ENDPROGRAM
```

### 4.3.2 Implementação do configurador com o uso de um sistema especialista

A conceituação de um sistema especialista justifica seu uso em casos onde se verifique [43]:

- a existência de um especialista humano no assunto que se deseja resolver
- o conhecimento sobre o problema descrito conceitualmente
- a necessidade do conhecimento em ambiente distintos
- a utilização de regras heurísticas na resolução do problema

No caso específico da configuração de um núcleo, a primeira justificativa é satisfeita, uma vez que o levantamento das características de uma determinada aplicação implica naturalmente na existência de um especialista que conhece o perfil do núcleo para ela. O segundo item é satisfeito, uma vez que a teoria de sistemas operacionais é bastante detalhada. O terceiro ítem é satisfeito ao ampliar-se o alcance de tal trabalho (núcleo configurável) em várias aplicações. São utilizadas regras heurísticas na configuração quando se confrontam itens como performance, características específicas de equipamentos, e outros, com as necessidades da aplicação. Ainda se verifica neste tópico a utilização de heurísticas na resolução das inter-relações entre os elementos do núcleo e na definição do que pode ser considerado um núcleo mínimo ou ideal, dada uma aplicação.

O programa configurador é, por si próprio, o sistema especialista. A escolha de uma ferramenta para construção do sistema especialista deve levar em conta o grau de liberdade que se deseja obter na implementação dos mecanismos de inferência sobre os dados do programa configurador. Uma ferramenta mais sofisticada certamente não dará ao programador flexibilidade para resolver problemas específicos de sua aplicação. Para a escolha de uma ferramenta deve-se levar em conta o quanto é viável a representação do problema pela ferramenta, qual o suporte para implementação e depuração existentes, e a confiabilidade da mesma. A ferramenta escolhida deve ser adequada para que se possa ter um primeiro protótipo rapidamente e interativamente avaliar e incrementar o comportamento do sistema.

Uma ferramenta adequada deveria possibilitar a representação de características do núcleo, ter uma interface clara com o utilizador e meios de integração com o sistema operacional da máquina hospedeira, de forma a produzir o núcleo final sem ônus do implementador com questões como geração de código final.

Há várias ferramentas possíveis, desde aquelas que apresentam a interface com usuário e máquina de inferências já prontas até as que devem ser totalmente construídas via linguagens convencionais ou com linguagens já voltadas para inteligência artificial, como Prolog.

Uma vez que o programa configurador não é de utilização intensiva, pode ser mais interessante utilizar ferramentas o mais sofisticadas possíveis, de maneira a incluir apenas as regras que especificam o núcleo e ter o processo de interface com o usuário e a máquina de inferências prontas do que implementar, por exemplo, em Prolog, o mesmo programa. Em qualquer caso, é importante que a ferramenta se adeqüe aos requisitos descritos anteriormente.

Para expressar o conhecimento necessário sobre o núcleo deve-se construir uma estrutura de dados que descreva ítens e características dos módulos componentes do núcleo. Algumas partes desta estrutura são estáticas, contendo informações fixas sobre o núcleo, e outras partes dinâmicas, criadas para uma solução específica do problema em questão. Sobre esta base de dados opera-se através de regras; uma regra contém uma ou mais condições que devem ser satisfeitas no contexto descrito pelos dados. Uma regra pode alterar a base de dados, e assim levar a situações distintas em problemas distintos, frente a uma seleção particular do usuário, ou apenas pode verificar uma condição. As regras vão representar inferências lógicas ou heurísticas. A estratégia com a qual se aplicam as regras sobre os dados deve ser voltada para a obtenção final de um núcleo determinado, adequado às solicitações do usuário.

As regras sobre o núcleo devem espelhar as dependências entre seus elementos, a formação de um núcleo mínimo, as seleções prováveis para obtenção de características desejadas, como performance e tamanho. Ainda deve ser controlada a duplicação de serviços semelhantes e as quantidades relacionadas a serviços do núcleo.

Partindo do discutido no item anterior, é possível que se tenha uma noção real de como vai ser construído e utilizado o sistema especialista para configuração do núcleo. A operação do sistema pode se dar em vários sentidos. Uma utilização possível é a da seleção de serviços apropriados para determinado tipo de núcleo. Para isso, a base de dados deve conter todas as informações sobre cada serviço, suas necessidades para implementação e seus relacionamentos com os demais . Outra utilização possível é a partir de um certo tipo de aplicação, onde a base de dados deve ter as características de forma a que, dados os requisitos daquela, o configurador saiba sugerir um

conjunto de serviços ou atributos destes.

A base de dados que expressa estas relações pode ser escrita na forma de regras, sendo, por exemplo, entrada de uma ferramenta de construção de sistemas especialistas. A forma de representação seria então em um esquema *"SE ENTÃO SENÃO"*.

### Exemplo de seleção de serviços

Um dos serviços possíveis em um núcleo é o de temporização. As atribuições da temporização são:

- manter registro de quantidades determinísticas de tempo
- fornecer meios de controle de inatividade de entidades, devido a espera de eventos, e com isso evitar situações de "dead-lock"
- controlar tempos em geral, por exemplo para monitorar entidades do mundo externo

Estes tipos de serviço podem não existir no núcleo, de acordo com as necessidades deste. Um núcleo que pretenda apenas fornecer um ambiente multi-tarefas a processos "bem-comportados", que não tomem indefinidamente o processador, não necessita deste controle de temporização. Um núcleo com requisitos críticos de tempo com certeza precisa deste serviço.

Para sua implementação, o serviço de temporização se vale de entidades básicas ou essenciais. Quando de sua existência, permite a agregação de outros serviços que o utilizem, total ou complementarmente. Em uma abordagem simples, seja o esquema de relações do serviço de temporização como na figura 4.3.2 - pag. 79:

A partir destas relações um esquema da base de dados seria como segue:

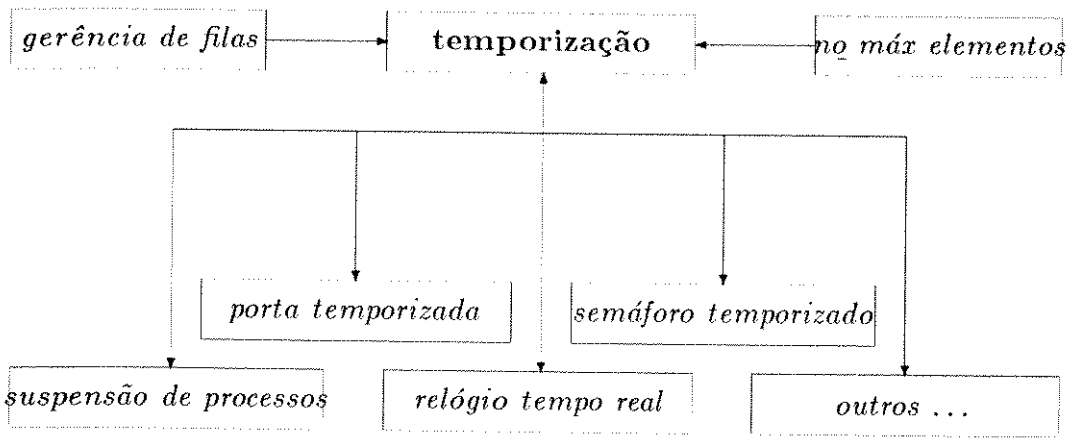



Figura 4.1: Relações do serviço de temporização

```
SE    temporizacao E' definida
E     gerencia de filas E' NAO definida
ENTAO solucao E' erro
```

```
SE    temporizacao E' definida
ENTAO semaforo temporizado E' existente
```

```
SE    temporizacao E' definida
ENTAO porta temporizada E' existente
```

```
SE    temporizacao E' definida
ENTAO relogio de tempo real E' existente
```

```
SE    temporizacao E' definida
ENTAO suspensao de processo condicionado a tempo  E' existente
```

```
SE    temporizacao E' definida
ENTAO numero de celulas E'  valor maximo de temporizacao
```

```
SE    valor maximo de temporizacao E' definido
ENTAO valor maximo de temporizacao E' 10 OU 20 OU 30
```

```
SE    nucleo definido E' com temporizacao
E     temporizacao E' NAO definida
ENTAO nucleo final E' errado
```

### Exemplo de seleção de tipos de núcleo

Outro serviço oferecido pelo configurador seria, a partir dos requisitos de uma aplicação, sugerir os melhores conjuntos de funções. Para isso, o sistema especialista poderia se valer também de conhecimentos sobre conjuntos já formados, que poderiam fazer parte da base de conhecimento. As opções mais adequadas podem ser apresentadas em forma de percentagem, deixando à aplicação a seleção final.

A base de conhecimento deveria conter para isso modelos já prontos

de núcleo, e permitir que a partir destes modelos uma ou outra característica seja alterada, de maneira a obter um resultado final mais rapidamente.

A sugestão das características dos componentes de um núcleo a partir de seus requisitos é outra possibilidade de atividade do configurador. Um exemplo deste tipo de operação do configurador do núcleo pode ser visto na figura 4.2 - pag. 81.

| *Núcleo* | *Escalonador* | *Sincronização* | *Comunicação* |
|---|---|---|---|
| tempo crítico | preemptivo | semáforos | memória partilhada |
| tarefas pequenas | não-preemptivo | sinais | sinais |
| muitas interrupções externas | preemptivo | portas | portas |

Figura 4.2: Exemplo de formações de núcleos

Neste caso, as regras, seguindo o esquema apresentado no item anterior, seriam:

```
SE nucleo E' com tempo critico
ENTAO escalonador E' preemptivo

SE nucleo E' com tempo critico
ENTAO sincronizacao E' semaforo

SE nucleo E' com tempo critico
ENTAO comunicacao E' memoria partilhada

SE nucleo E' com tempo critico
ENTAO temporizacao E' sim
```

```
SE nucleo E' com tarefas pequenas
ENTAO escalonador E' nao-preemptivo

SE nucleo E' com tarefas pequenas
ENTAO sincronizacao E' sinais

SE nucleo E' com tarefas pequenas
ENTAO comunicacao E' sinais

SE nucleo E' com tarefas pequenas
ENTAO temporizacao E' sim

SE nucleo E' com muitas interrupcoes externas
ENTAO escalonador E' preemptivo

SE nucleo E' com muitas interrupcoes externas
ENTAO sincronizacao E' portas

SE nucleo E' com muitas interrupcoes externas
ENTAO comunicacao E' portas

SE nucleo E' com muitas interrupcoes externas
ENTAO temporizacao E' sim
```

### 4.3.3 Considerações sobre programas de configuração

O ambiente sobre o qual se apoia o configurador deve possuir os módulos fontes contendo as funções que compõem o núcleo, espaço

para o armazenamento de dados gerados durante o processo de configuração, bem como para os processos subseqüentes de compilação, ligação e geração de biblioteca.

O diálogo entre o programa e o usuário deve ser de fácil compreensão e de visualização agradável, sugerindo o uso de janelas, cores, movimentação de cursor entre opções, dependendo dos recursos disponíveis. Uma opção interessante é o fornecimento de um "help" detalhado para cada ítem configurável, tornando o sistema utilizável por vários projetistas e não somente o "gerente" do desenvolvimento.

Observa-se que existe, em qualquer caso, uma utilização intensiva de manipulação de textos; o uso de ferramentas como geradores de "parsing" e analisadores léxicos pode ser de auxílio nestas ações, principalmente na fase de geração da base de dados, tanto em uma quanto em outra filosofia de implementação.

O programa de configuração deve ser estruturado de forma a acompanhar as evoluções do núcleo, como por exemplo controlando a geração para máquinas e configurações hardware distintas, ou produzindo uma bateria de testes adequada para o núcleo por ele gerado.

# Capítulo 5

# O exercício de implementação de um núcleo segundo o modelo de entidades

Neste capítulo serão discutidas as formas de utilização do núcleo implementado segundo o modelo proposto, tanto do ponto de vista de alteração do código, quanto o de utilizá-lo para uma aplicação. As opções desta implementação serão discutidas e criticadas posteriormente.

## 5.1 Utilização do núcleo

Do ponto de vista de utilização existem duas formas de trabalho:

- utilizar funções já existentes
- modificar as funções existentes, onde se inclui o acréscimo de novas funções

O núcleo é composto, como código, de vários módulos que formam uma biblioteca. Esta biblioteca é ligada ao código da aplicação, fornecendo as funções necessárias. Do ponto de vista apresentado no modelo tem-se as seguintes características apresentadas a seguir.

### 5.1.1 Entidade essencial

- gerência de interface com o hardware
  É um código dependente de máquina, particular para a máquina utilizada (IBM-PC) e, para fins de otimização, são escritos, em sua maioria, em linguagem de montagem ("assembly"). Este código engloba as funções que agem sobre o esquema de controle de tempo da máquina, sobre as ações para tratamento de interrupções e ainda sobre a execução da troca de contexto entre tarefas. Ainda inclui-se nesta categoria funções de caráter específico do ambiente em uso, como da modificação dos testes

para verificação de pilha, colocados pelo compilador. Como há uma pilha para cada tarefa, a execução da função que testa a pilha, provida pelo compilador, leva a erro ("stack overflow"). Para isso não acontecer desabilita-se este teste, ou acionando uma flag do compilador ou modificando o código gerado por ele. Estas ações são invisíveis ao utilizador do núcleo. A troca de contexto é efetuada por um conjunto de rotinas que preparam o ambiente para a próxima tarefa, e armazenam o ambiente da tarefa atual. Para otimização de código estas rotinas são estruturadas para se interligarem fora dos padrões normais; existem nesta implementação rotinas que transferem o controle para outras diretamente. Incluem-se aqui funções que iniciam e terminam a operação multitarefa.

Alterações nos módulos desta gerência podem ser feitas para adequá-lo a outros modelos de memória (estão implementadas as interfaces para os modelos "large" e "small"), para adaptá-lo a outros compiladores ou para fins de otimização. Também podem ser incluídos módulos para tratamento de outras interrupções ou de erros hardware.

Na tabela 5.1 - pag. 87 estão relacionadas as funções desta gerência e as rotinas que as implementam (fornecedoras de serviço). Os algorítmos das rotinas estão descritos no apêndice 2.

| função | rotina | módulo |
|---|---|---|
| controle de tempo | prog_rel<br>desprog_rel | schedule.asm |
| tratamento e controle de interrupções | serv<br>cli<br>sti<br>block<br>release | schedule.asm |
| início e terminação | ini_esc<br>fim_esc | swap.asm |
| operação multitarefa | inicia_operacao | gint.c |
| ação sobre tarefas | instal tar<br>swap_in | swap.asm |
| controle de pilha | altera_chk_pilha<br>reseta_chk_pilha<br>mychktsk | swap.asm |

Tabela 5.1: Gerência de interface com o hardware

– gerência de escalação de tarefas

São os serviços que criam e supervisionam as tarefas no núcleo. Nesta implementação esta gerência é dividida em 2 unidades de compilação, uma que controla as tarefas do núcleo e outra que faz a lógica para troca de contexto. Esta última é "chamada" pela rotina acionada pela interrupção de relógio (rotina SERV). Esta rotina não é acessada pela aplicação; no caso de necessitar-se de outra filosofia para seleção de tarefas para execução, apenas se trocaria a rotina de escalonamento. No caso desta implementação, a política de escalonamento baseia-se em prioridades, tendo apenas uma fila de tarefas ativas ordenada por estas prioridades, que são definidas quando da criação da tarefa.

Na tabela 5.2 - pag. 89 estão relacionadas as funções às rotinas que implementam os serviços desta gerência. Seus algorítmos encontram-se no anexo A - pag. 130.

| função | rotina | módulo |
|---|---|---|
| criação de tarefas | cria_tarefas | proc.c |
| destruição de tarefas | destroi_tarefa | proc.c |
| reorganiza fila de ocupação do processador | reescalona | proc.c |
| coloca tarefa em "suspenso" | suspende_tarefa | proc.c |
| retira tarefa de "suspenso" | resume_tarefa | proc.c |
| altera prioridade da tarefa | muda_pri | proc.c |
| controla condição para chaveamento de contexto | escalonador | escala.c |
| controle da fila de tarefas do escalonador | busca proc<br>del_queue<br>pq_cmp<br>pq_swap | proc.c |

Tabela 5.2: Gerência de escalação de tarefas

– gerência de sincronização e comunicação entre tarefas
São implementados serviços que controlam listas de elementos, tratados aqui como "portas". Existe uma única unidade de compilação que engloba todos os serviços desta gerência.

Na tabela 5.3 - pag. 91 estão relacionadas as funções às rotinas que implementam os serviços desta gerência. Seus algorítmos encontram-se no anexo A - pag. 130.

| função | rotina | módulo |
|---|---|---|
| criação de portas | cria_porta | porta.c |
| libera porta | libera_porta | porta.c |
| espera mensagem em porta | espera_mens | porta.c |
| consome mensagem em porta | consome_mens | porta.c |
| obtém mensagem em porta | obtem_mensagem | porta.c |
| envia mensagem para porta | envia_mens | porta.c |

Tabela 5.3: Gerência e comunicação e sincronização

### 5.1.2 Entidade básica

O código desta entidade é responsável pelos serviços "genéricos" do núcleo. Para cada gerência implementada - filas e memória - existe uma unidade de compilação.

A implementação de filas utiliza um esquema de ordenação por "heap". A gerência de memória controla os apontadores devolvidos pelos servi c cos da entidade essencial (que são as próprias rotinas da biblioteca "C"). O controle destes apontadores é feito através da manutenção de uma tabela (com os apontadores) ordenada em valores ascendentes; é utilizado um algorítmo de busca binária para este controle.

A inclusão de novas gerências nesta entidade, como de calendário e E/S, deve seguir o mesmo esquema (unidade de compilação = gerência).

Na tabela 5.1 - pag. 93 estão relacionadas as funções às rotinas que implementam os serviços desta gerência. Seus algorítmos encontram-se no anexo A - pag. 130.

| função | rotina | módulo |
|---|---|---|
| rotinas auxiliares | reheap_down<br>reheap_up | pq.c |
| criação de uma fila | cria_fila | pq.c |
| insere elemento em fila | ins_fila | pq.c |
| retira elementos da fila | del_fila | pq.c |
| troca lugar de elementos na fila | troca_fila | pq.c |
| remove elementos da fila | remove_fila | pq.c |
| verifica elemento no topo da fila | look | pq.c |
| informa número de de elementos na fila | numele | pq.c |
| solicita quantidade de memória | pede mem | mem.c |
| libera memória | libera_mem | mem.c |
| informa quantidade de memória disponível | mem_disp | mem.c |
| controle de memória | dmalloc<br>dfree<br>remove from table<br>add_to_table<br>find_in_table | mem.c |

Figura 5.1: Gerências da entidade básica

### 5.1.3 Entidade de serviço à aplicação

Nesta implementação esta entidade existe como uma definição funcional do núcleo, e não como um módulo compilável. No anexo B - pag. 180 estão descritas, separadas por gerências e entidades, as funções do núcleo. É apresentado o conjunto de parâmetros de entrada, saída, erros e uma breve descrição do funcionamento.

Uma outra implementação poderia ver esta entidade como uma "shell" sobre um núcleo; neste caso, haveria uma ou mais unidades de compilação para este fim. Novos serviços deveriam estar em unidades de compilação separadas, preservando a possibilidade de configuração.

### 5.1.4 Exemplo da modificação de um núcleo

Há uma divisão entre serviços essenciais, básicos e de fornecimento de funções dentro do modelo utilizado. Sob esta óptica, suponha-se que seja necessário que se façam algumas modificações no núcleo, por exemplo na política de escalonamento e para inclusão de uma nova função.

A nova política de escalonamento pode ser um "round-robin"/"time-sharing" simples, onde haveria uma só fila de processos à espera de execução, sem prioridade senão aquela expressa na posição do mesmo nesta fila.

Seja a nova função um "pipeline", necessário para comunicação de uma quantidade contínua de dados. A primeira atitude é estudar a ação desta função dentro código já existente, através de sua classificação no modelo definido; depois, especificar estes serviços funcionalmente. No caso, esta função comporia uma nova gerência, inserida na entidade de serviço à aplicação.

Os ítens de especificação funcional e interface para utilização para as duas modificações sugeridas são apresentados a seguir.

### Nova política de escalonamento

Seja uma política de escalonamento, "round-robin", que executa uma tarefa em seguida a outra, na ordem em que são criadas, ciclicamente. Uma característica "time-sharing" implica que a tempos fixos o controle do processador passe de uma tarefa para outra. Assim, é de interesse que a fila de processos prontos para execução mantenha ordenação espelhando a ordem de criação e colocando uma tarefa ao fim de sua execução no fim da fila de "prontos". Para a ação de "time-sharing" a rotina que faz o escalonamento deve verificar, antes de proceder à sua ação, se o tempo de execução do processo corrente já venceu.

Para que a fila seja ordenada pela ordem de criação basta fazer com que a rotina de comparação entre elementos da fila retorne 0 (não há chave de ordenação senão a própria inclusão na fila quando da criação do processo). Para que se faça a verificação do "time-sharing" é necessário definir uma variável que armazene o tempo de execução de cada processo, e uma constante para o tempo máxima de execução. Esta variável é zerada quando da escalação do processo, é incrementada a cada ocorrência de interrupção e testada com a constante de tempo para execução. Quando coincidente com o valor máximo é executada a troca de contexto, novamente zerada a variável, ocorrendo então ação semelhante para a nova tarefa.

Não existe modificação na interface para o usuário, uma vez que a ação de chaveamento entre processos é invisível à aplicação. Para configuração, utiliza-se nova unidade de compilação com o "nome

padrão" *ESCALA*.

### Novo serviço - Pipeline

Apesar da existência de mensagens para intercomunicação entre tarefas, um "pipeline" pode ser necessário para os casos onde existe envio de um fluxo contínuo de dados entre elementos. Um "pipeline" [44] pode ser visto como um "tubo" que é aberto nas extremidades simultaneamente e onde se efetuam tanto leituras quanto escritas. Visto como uma quantidade de informação restrita a uma área, o "pipe" sofre escritas até que se complete o espaço reservado a ele, e então o tarefa que faz a escrita tem bloqueada a sua execução; outro tarefa pode fazer leituras consumindo os dados desta área, e então, quando os mesmos se esgotam, a mesma é bloqueada. Apenas dois tarefas se utilizam de um mesmo "pipe" ao mesmo tempo, não necessariamente os criadores do mesmo e também não os mesmos durante toda sua existência no núcleo. Um "pipe" não é ligado em particular a nenhum tarefa. É aberto por nome, e pode ser usado indistintamente. A implementação é na forma de um buffer circular, com bloqueio controlado para leitura e escrita.

O módulo que implementa a função de "pipe" deve englobar todas as funções necessárias ao funcionamento do mesmo, inclusive as estruturas que o definem. É classificado na categoria de entidade de serviço à aplicação e compõe uma nova gerência.

### Pipes - Interface Funcional

- *Cria pipe*
- **função:**

  Aloca um pipe para envio de dados

- **formato de chamada:**

  int cria_pipe(char *nome_pipe,unsigned tam_pipe,PIPE &id_pipe);

- **parâmetros de entrada:**

  * nome_pipe: nome do pipe a ser criado
  * tam_pipe: quantidade de dados que o pipe pode conter

- **parâmetros de saída:**

  * NO_MEM;
  * OPERAÇÃO OK, e "id_pipe" contém a identificação (interna) do pipe criado.

- **descrição:**

  É chamada por um tarefa que deseja enviar dados a outro, que não precisa existir neste momento ainda. A função verifica a existência do pipe selecionado por "nome_pipe". Caso exista, conecta a tarefa em execução com a outra que solicitou anteriormente a abertura do "pipe", colocando esta última como ativa. Caso não exista o "pipe", cria estrutura de controle para ação posterior semelhante a descrita acima, e o tarefa solicitante é bloqueado até que algum receptor se conecte ao "pipe" criado. Há um tempo máxima, selecionado por configuração, para que dure esta espera.

- **erros possíveis:**

  Falta de recursos (não existência de espaço para alocação de um novo "pipe").

- *Conecta pipe*

- **função:**

  Solicita dados de um "pipe".

- **formato de chamada:**

  int conecta_pipe(char *nome_pipe,PIPE &id_pipe);

- **parâmetros de entrada:**

  * nome_pipe: nome do pipe a quem o tarefa corrente se conecta

- **parâmetros de saída:**

  * NO_MEM
  * OPERAÇÃO_OK, e "id_pipe" contém a identificação (interna) do pipe referenciado.

- **descrição:** Um tarefa que deseja receber dados usa esta função para ligar-se a um "pipe" já criado ou não. Caso não exista o "pipe" a tarefa solicitante é bloqueada. É a função dual de "cria_pipe".

- **erros possíveis:**

  Falta de recursos (não existência de espaço para alocação de um novo "pipe").

- *Deleta_pipe*

- **função:**

  Termina a existência de um "pipe" no núcleo

- **formato de chamada:**

  int deleta_pipe(PIPE *id_pipe);

- **parâmetros de entrada:**

  * id_pipe: identificação interna do "pipe" a ser deletado

- **parâmetros de saída:**

  * PIPE_OCUPADO

* OPERAÇÃO_OK, e "id_pipe" contém a identificação (interna) do pipe referenciado.

- **descrição:** A função deleta um "pipe" criado anteriormente. Deve ser chamada pelo receptor de dados, ao fim de sua última leitura. Efetua a liberação da memória alocada para o "pipe".
- **erros possíveis:**

  Erro se houver dados a serem lidos no "pipe".

- *Le_pipe*
- **função:**

  Lê um dado no "pipe"

- **formato de chamada:**

  int le_pipe(PIPE *id_pipe,unsigned n_bytes, void *dest_data);

- **parâmetros de entrada:**
  * id_pipe: identificação interna do "pipe" utilizado
  * n_bytes: número de bytes a serem lidos
  * dest_dados: apontador para área de destino dos dados lidos
- **parâmetros de saída:**
  * PAR_INCORRETO
  * OPERAÇÃO_OK
- **descrição:** A função controla a leitura em um "pipe" de um número especificado de bytes. Controla a ação sobre o buffer circular sobre o qual é especificado o "pipe".
- **erros possíveis:**

  Erro se não existe o "pipe" especificado.

- *Escreve_pipe*

- **função:**

  Escreve um dado em um "pipe"

- **formato de chamada:**

  int escreve_pipe(PIPE *id_pipe,unsigned n_bytes, void *org_data);

- **parâmetros de entrada:**

  * id_pipe: identificação interna do "pipe" utilizado
  * n_bytes: número de bytes a serem escritos
  * dest_dados: apontador para área de origem dos dados

- **parâmetros de saída:**

  * PAR_INCORRETO
  * OPERAÇÃO_OK

- **descrição:** A função promove a escrita de dados no "pipe"; para isso, transfere-os para o buffer circular sobre o qual o "pipe" se baseia.

- **erros possíveis:**

  Erro se não existe o "pipe" especificado.

## 5.2 Utilização do núcleo do ponto de vista da aplicação

O núcleo apresenta-se à aplicação como uma biblioteca de rotinas capaz de prover funções que fornecem a um ambiente de programação características multi-tarefas. A aplicação não enxerga os valores utilizados internamente pelas rotinas do núcleo. Sendo assim, eventuais mudanças na biblioteca, como as sugeridas no ítem anterior, não se

refletem sobre os programas já prontos, a menos de conceitos formais. Este ítem visa a apresentar um guia para utilização do núcleo conforme esta implementação.

Para utilização do ambiente multi-tarefa a aplicação deve se estruturar de maneira conveniente; deve definir tarefas e as portas de comunicação entre elas. O projetista da aplicação deve saber definir quais são as partes de seu código que configuram tarefas e a forma de comunicação entre elas.

O conceito de tarefa - uma subrotina que contém sua própria pilha e conjunto de valores de registradores - deve ser diferenciado do de um procedimento convencional. O que define uma tarefa é o ambiente em que ela é executada, e não somente seu código. Daí vem o conceito de código reentrante, que significa o compartilhamento do mesmo código em ambientes distintos. Cabe recordar que este código não deve ter variáveis "static", ou seja, que podem ser alteradas por todas instâncias do código. Uma vez que cada tarefa tem sua própria pilha, e porque o controle da execução do programa (o "program counter") é também parte do ambiente da tarefa, é possível a definição de variáveis locais (armazenadas sobre a pilha da tarefa) e cada tarefa se "lembra" do seu ponto de execução quando retoma o controle do processador.

Para o programador da aplicação é necessário manter em mente que tarefas não são subrotinas, e que portanto não se pode passar informações entre elas através de argumentos e valores de retorno. A comunicação entre tarefas é feita somente por "portas" ou variáveis globais, estas não controladas pelo núcleo.

Os resultados de ações do núcleo são reportados ao usuário através de vários códigos de erro. A aplicação não deve usar nomes coincidentes com as rotinas de uso interno do núcleo.

### 5.2.1 Operação Multitarefa

Um programa normal deve criar tarefas e depois iniciar a operação multitarefa. Não é necessário que todas as tarefas sejam criadas "a priori", uma vez que tarefas podem criar tarefas.

O início da operação multitarefa se dá com uma chamada à rotina "inicia_operação". Esta rotina trata da alteração do relógio do sistema, utilizando para isso um parâmetro passado à rotina. Este parâmetro representa em quantas vezes se altera a freqüência de interrupção do relógio da máquina (o PC sofre aproximadamente 18,2 interrupções por segundo).

Foi utilizada uma programação de maneira a alterar o relógio original do PC de valores inteiros, de forma a manter o controle sobre as operações dependentes de tempo antes do início da atividade do núcleo.

No PC, o "timer-chip" - 8253 - é programado pelo MS-DOS, na partida, para gerar interrupções a cada 54,9 milisegundos, o que é o maior intervalo de tempo possível. O menor intervalo entre interrupções de relógio é obtido através da programação para uma contagem de 65536 "ticks" do relógio. Qualquer outro valor de contagem deve ser inteiro e divisor deste valor; daí o fato do parâmetro "fator" ser uma potência de 2 [45].

Por exemplo:

```
interrupcao a cada 14 mseg

fator mais proximo = 4
```

portanto,

```
valor de contagem = 65536 / 4 = 16384

tempo real entre interrupcoes = 13,73 mseg
```

O controle passa imediatamente à tarefa mais prioritária quando do início da operação multitarefa.

O retorno da função "inicia_operação" se dá em dois casos:

- quando é deletada a última tarefa do sistema
- a uma chamada para a rotina de fim de operação ("fim_operação")

No caso de retorno normal, é garantida que toda a memória alocada para o controle da tarefa foi devolvida ao sistema.

O fim da operação multitarefa se dá através da chamada à função "fim_operação". Esta função "desliga" a operação multitarefa e o controle passa para a rotina que havia chamado "inicia_operação", exatamente após a chamada em questão. Existe assim um retorno forçado de "inicia_operação". Um código de erro é associado a esta ação e é passado para a rotina chamadora como o valor de retorno (simulado) desta rotina.

### Operação sobre Tarefas

A criação de tarefas é feita através de uma chamada à função "cria tarefa". A chamada a esta função força um reescalonamento. Ou seja, se a

tarefa criada for de maior prioridade que a atual, o controle passará à tarefa criada.

A prioridade de uma tarefa pode ser alterada a qualquer momento, inclusive pela própria, através da função "muda_pri". Este serviço força a ocorrência de um reescalonamento.

Uma tarefa é deletada através da função "destrói_tarefa", que libera a memória alocada para o controle da tarefa. A memória alocada pela tarefa, durante sua existência, deve ser liberada explicitamente pela própria tarefa, antes de sua extinção. A função de deleção de tarefa força um reescalonamento.

A função "reescalona" força uma tarefa de maior prioridade, ainda suspensa, tomar o processador.

Durante a operação da aplicação pode ser necessário que uma tarefa espere por alguma ação a ocorrer ou então tenha que ceder o uso do processador. Para que se dê esta espera são fornecidas as funções "suspende_tarefa", que suspende a tarefa ativa por um tempo determinado na chamada, e "resume_tarefa", que ativa uma tarefa suspensa antes do decorrer do tempo especificado quando de sua suspensão.

### 5.2.2 Comunicação entre Tarefas

A comunicação entre tarefas se dá via portas. A criação de uma porta se dá através da função "cria_porta". Há para uma porta um número máximo de mensagens que podem estar na mesma simultaneamente. Não há limite para o número de tarefas que podem pender em uma porta à espera de mensagem.

Durante o tratamento da interrupção do relógio é feita uma varredura em todas as portas existentes, a fim de detectar-se "timeouts".

Esta busca é onerosa em relação a tempo; o projetista da aplicação deve criar portas de maneira a "balancear" os tempos de busca no tratamento do relógio.

O envio de mensagens a uma porta se dá via "envia_mens". O que é armazenado na porta é um ponteiro para a mensagem, que pode ser de qualquer tipo. A mensagem propriamente dita deve ser declarada na memória estática. Se não há nenhuma tarefa à espera de mensagem, esta função retorna imediatamente; caso contrário, a mensagem é entregue à tarefa à espera, que é colocada no estado "ativo", provocando um reescalonamento no núcleo.

A recepção de mensagens se dá através de "espera_mens" e de "obtém mens". A primeira função tem dois modos de operação: ela pode apenas verificar a existência de mensagens na porta especificada ou então solicitar explicitamente uma mensagem . Neste caso, se a mensagem não estiver presente, a função provocará a suspensão da tarefa solicitante por um tempo máximo determinado por configuração.

Se for efetuada apenas a verificação da existência de mensagem, a obtenção da mesma pode ser feita via "obtém_mens".

Os pedidos de mensagem que uma porta recebe são enfileirados na ordem de chegada, não valendo prioridades nesta espera. Isto se deve a dois fatores:

- simplicidade
- estrutura da maioria das aplicações, onde tarefas de mesma prioridade não ficam à espera de mensagens na mesma fila

Se a mensagem solicitada está presente, "espera_mens" (com opção de consumo) a entrega imediatamente ao solicitante; caso contrário, a tarefa é suspensa e ocorre um reescalonamento.

Um erro possível nestas operações é o "timeout", que ocorre no vencimento do tempo de espera sem chegada de mensagem.

### 5.2.3 Controle de Concorrência

São fornecidas funções pela gerência de interrupções para controlar o acesso ao processador. As funções para desabilitar e habilitar interrupções ("bloqueia_interrupção" e "libera_interrupção") devem ser usadas com cuidado. As funções para bloqueio/liberação do escalonador são utilizadas para controlar o acesso ao processador. A função "bloqueia_escalonador" desabilita a operação do escalonador mas não a interrupção de relógio. A tarefa que está em execução não perde o controle do processador; entretanto, interrupções que venham a ocorrer não serão tratadas. A função dual é "libera_escalonador".

## 5.3 Considerações sobre a implementação

O modelo apresentado no capítulo 1, e as discussões sobre características e estratégias de implementação, foram reunidas para a construção de um pequeno núcleo. A máquina alvo foi um IBM-PC, e a linguagem de implementação utilizada foi "C", com algumas rotinas específicas em "assembly". Seu primeiro uso foi para o fornecimento de um ambiente de execução adequado ao projeto SISDI-MAP [46], [47], [48].

Os conceitos do modelo foram utilizados para localizar as funções em "gerências"; uma gerência foi mapeada em um a unidade de compilação "C". Com isso, obteve-se o encapsulamento necessário à configuração funcional ("macroscópica"). Foram isolados em unidades

de compilação os serviços configuráveis a nível de algorítmos ( configuração "microscópica").

A configuração de valores de elementos do núcleo, como tempos máximos de espera, intervalos de prioridades, quantidade de tarefas e mensagens por portas, foi mapeada de duas formas: uma onde se definiu um arquivo ".h" com os valores configuráveis, e outra onde os valores são obtidos a partir de um arquivo texto em disco, antes da iniciação da operação multitarefa. A primeira forma implica em que haja uma compilação de todos os módulos em que os valores são utilizados, o que não acontece na segunda forma.

Para implementação deste núcleo foram utilizadas rotinas de domínio público [49], [50], [32], [51], [45], [52], [44]. Uma interface "padrão", definida a partir de [53], [54], faz as vezes de "entidade de serviço à aplicação". Esta interface é apresentada no anexo B - pag. 180; para cada função de uma gerência são descritos o formato de chamada, parâmetros de entrada e de saída e os erros que podem decorrer de sua utilização. Optou-se por um estilo de interface onde parâmetros de saída são retornados em variáveis definidas pelo chamador, e erros como valores de retorno. A razão para isso é a devolução de um único tipo de valor ("erro") na chamada de uma função do núcleo, havendo assim uma interface padrão entre rotinas da entidade e o código da aplicação.

As funções de caráter genérico formam a entidade básica; fazem parte desta entidade as gerências de fila e de memória, cada qual equivalendo a uma unidade de compilação.

Rotinas para programação de hardware, para gerência das tarefas e da comunicação entre elas formam a entidade essencial. Os serviços de interface com hardware, escritos em assembly 8086, localizam-se em módulos de compilação separados. Faz parte desta entidade um

módulo com a definição de variáveis globais, e outro, utilizado pela aplicação, que contém as definições de tipos globais e interfaces às funções do núcleo, conforme a entidade de serviço à aplicação.

O núcleo é fornecido à aplicação como uma biblioteca; é ligado às rotinas da aplicação, gerando-se um único código executável.

### 5.3.1 Opções de implementação

Durante a implementação do núcleo manteve-se como princípios básicos a estruturação segundo o modelo definido e a utilização de rotinas de domínio público. A estruturação em entidades/gerências/serviços visa essencialmente permitir a configuração do núcleo; a utilização de rotinas prontas procura se valer da reusabilidade de idéias genéricas de programação, com economia de desenvolvimento e de testes dos algorítmos.

Como a primeira utilização do núcleo não se daria em um ambiente "tempo real", não foi preocupação durante a implementação otimizar código. Mesmo a filosofia de escalonamento escolhida é criticável: como pode ser observado no apêndice 2, a ocorrência de uma interrupção de relógio vai provocar troca de contexto para a execução da tarefa mais prioritária no estado "ativo". Podem então haver situações em que tarefas de baixa prioridade tenham tempos muito curtos para serem executadas. A aplicação deveria adequar o intervalo de prioridades entre tarefas e utilizar mecanismos como a suspensão por um certo tempo de uma tarefa, de forma a garantir a execução de tarefas com baixa prioridade.

Outra opção discutível é a da gerência de comunicação e sincronização. Foi implementado um esquema de portas; a manutenção de mensagens é feita através da ordenação FIFO de seus apontado-

res, no próprio descritor (cabeçalho) da porta. As portas criadas têm tamanho fixo (quantidade de mensagens que podem receber), e são ordenadas conforma sua criação, em uma lista duplamente ligada. Outras implementações poderiam definir portas com prioridades e quantidades não limitadas de mensagens nas portas. Foi colocada uma característica de temporização nas portas: a cada interrupção de relógio todas são percorridas de modo a detectar um "timeout", condição sob a qual uma tarefa pendente na porta é colocada no estado "ativo". A manutenção das tarefas em espera nas portas não é ordenada; este é outra característica que poderia ser alterada em outra implementação. O tempo gasto durante a interrupção do relógio para detecção dos "timeouts" pode ser crítico em aplicações "real time"; certamente existem soluções mais adequadas, como o controle de apenas uma variável global que indicaria o "timeout" e a porta associada a ele.

A gerência de memória utilizou as funções pré-definidas na biblioteca da linguagem "C"; uma implementação em um ambiente com restrições de código poderia impedir a agregação da biblioteca "C" ao código executável.

O sistema operacional MS-DOS não foi utilizado; um gerenciador de recursos poderia ser implementado, a nível da entidade de serviço à aplicação, para acesso às funções DOS, uma vez que seus códigos não são adequados à operação multitarefa. O mesmo se aplica a operações de E/S.

Observa-se que as opções desta implementação seguem o modelo definido; alterações nos algorítmos não inviabilizam os demais nem afetam os usuários, desde que mantidas suas interfaces.

Foi utilizada aproximadamente a convenção de programação descrita na tabela 5.4 - pag. 110.

| modelo | implementação |
|---|---|
| serviço | rotina/unidade de compilação |
| gerência | unidade de compilação |
| entidade | conjunto de unidades de compilação |

Tabela 5.4: Relação implementação - modelo

O conjunto todo forma uma biblioteca; uma opção possível seria a do tratamento de cada entidade como uma biblioteca, permitindo a utilização em separado de cada uma, mediante controle do configurador.

# Capítulo 6

# Aplicação do modelo desenvolvido - implementação de protocolos de comunicação

A implementação de protocolos é uma das atividades onde é necessário um núcleo de tempo real como software básico. Isto se deve, em grande parte, à diversidade de ações tomadas por um protocolo e aos requisitos de tempo envolvidos nestas ações.

As áreas de interesse em comunicação têm procurado desenvolver software básico adequado às necessidades específicas de sua operação, como em [55], [56], [57], [58].

A construção de um núcleo, de forma a oferecer ao protocolo funções adequadas à sua implementação, pode ser guiada pelo modelo desenvolvido no decorrer deste trabalho. Para tanto, serviços específicos para esta implementação serão discutidos neste capítulo, dentro do contexto de entidades previsto pelo modelo.

## 6.1 Características de implementação de protocolos

O modelo OSI não referencia detalhes de implementação; com isso, aspectos que concernem ao software básico, como funcionalidade ou desempenho, não são discutidos. Observa-se contudo que existe uma separação clara entre software "aplicativo" e software "de comunicação", onde

- software "aplicativo": é entendido como o conjunto de funções que caracterizam a aplicação
- software "de comunicação": é aquele que fornece infra-estrutura para obtenção das informações necessárias à aplicação [59], [60]

### 6.1.1 Arquiteturas para implementação de protocolos

Na maioria da aplicações de redes de computadores, o software "aplicativo" reside em um computador hospedeiro onde existe um sistema operacional genérico, por exemplo um sistema operacional VMS ou UNIX. Isto se deve, em parte, ao "overhead" resultante da execução de todas as camadas de protocolo, previstas no modelo OSI, em um único processador [60]. O software aplicativo faz uso, normalmente, das funcionalidades destes sistemas operacionais genéricos e vê o suporte à comunicação como parte deste sistema operacional (por exemplo, acessando-o através de uma biblioteca).

Nesta arquitetura, o software "de comunicação" reside em um hardware especializado, como por exemplo uma placa ou um processador isolado [60], [61], [62], conectados ao computador hospedeiro da aplicação através de um barramento ou de uma via serial de alta velocidade. Neste cenário enquadra-se a discussão sobre a funcionalidade de um núcleo tempo real.

Um exemplo típico é apresentado em [60], onde observa-se a seguinte arquitetura (figura 6.1 - pag. 115):

- processador local
- memória de trabalho
- unidade de interface para a rede
- comunicação através de memória partilhada, tanto no sentido hospedeiro-placa quanto internamente à mesma

Outro exemplo de hardware de comunicação é apresentado em [61]. Explora-se, neste caso, uma estrutura multiprocessadora para tratamento da comunicação; existem módulos autônomos para tratamento

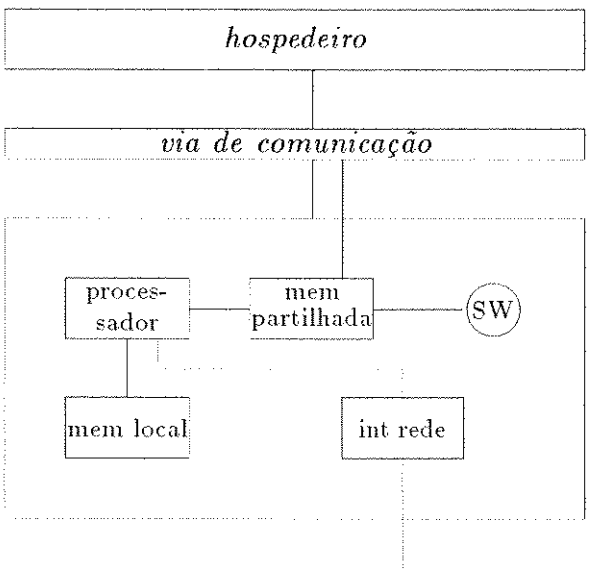



Figura 6.1: Modelo de implementação de sistema de comunicação

de conjuntos de funções dos protocolos, conectados por um barramento multinível.

### 6.1.2 Software para implementação de protocolos

O modelo OSI - camadas responsáveis por níveis de processamento, comunicação entre camadas através de mensagens - define implicitamente a estrutura do software para implementação de protocolos de comunicação. Esta estrutura é bastante hierarquizada (vide 6.2 - pag. 117), existindo uma "ordem" de execução entre camadas, comunicação entre camadas definidas, e um controle, a nível geral, do comportamento destas entidades.

Este tipo de estrutura sugere, para a organização do software, o mapeamento de camadas em tarefas e a comunicação por meio de portas dedicadas (unidirecionais). O controle de tarefas e sua intercomunicação são efetuados pelo software básico - ou seja, o núcleo de tempo real. Outra função deste núcleo é o fornecimento de insumos ao funcionamento das tarefas, como áreas de memória e temporizadores.

O núcleo assume, portanto, um papel de "gerente". Logo, adequar seus serviços à implementação dos protocolos significa definir funções e políticas de funcionamento condizentes com a estrutura software sugerida pelo modelo OSI.

Dentro do modelo desenvolvido neste trabalho, prevê-se que o núcleo seja configurável; com isso, o ajuste dos serviços pré-definidos pode ser feita tanto a um nível externo, com a alteração da filosofia de ação do serviço, quanto a nível interno, onde haveria um ajuste de parâmetros de execução do mesmo. É previsto no modelo que estas alterações ocorram sem prejuízo das demais entidades não modificadas.

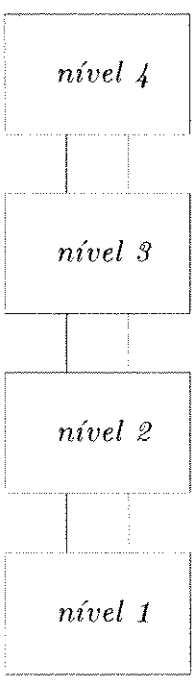



Figura 6.2: Estrutura hierárquica de protocolos

O estudo da implementação de protocolos deve fornecer insumos para esta configuração.

## 6.2 Principais questões em implementações de protocolos

### 6.2.1 Processamento de mensagens

A filosofia do processamento da mensagem é preocupação em todos os níveis de tratamento dentro de uma implementação de protocolos. Os diversos tipos de mensagens que trafegam entre as camadas dos protocolos levam à escolha de um esquema de "buffers" para seu processamento, principalmente no sentido de otimização no tratamento das mesmas.

Mesmo a níveis inferiores esta estrutura é prevista; o circuito integrado dedicado *MC 68824* Motorola-TBC [63], [64] possibilita uma organização desta forma. O integrado é configurado através de comandos que definem, em memória partilhada, modo de operação, teste do dispositivo, controle de modem, entre outras facilidades.

Para otimização da transferência de informação entre a rede e o processador dos níveis superiores, usa-se a mesma memória partilhada, estruturada de forma a conter "buffers" de dados interligados, configurados em relação a tamanho, conforme necessidade do usuário.

A estrutura de buffers é composta de um descritor de *frame*, descritores de buffer e buffers de informação. O descritor de *frame* contém informação de controle e atributos do *frame* e aponta para um descritor de buffer, que possui controles e informações sobre um buffer de dados que endereça, e que é o local efetivo da manutenção dos

dados de uma mensagem. Na necessidade de armazenamento de mais quantidade de informação, novos buffers de dados são alocados e seus descritores devidamente interligados, formando uma lista ligada. Não é necessário que a informação seja armazenada de forma contígua em memória.

Esta estrutura é exemplificada na figura 6.3 - pag. 120.

No descritor de buffer é mantido um contador que indexa, no buffer de dados, a posição de início dos dados úteis. Esta característica é útil na composição de mensagens; conforme a informação é passada entre os níveis de protocolo, retira-se ou acrescenta-se informação de controle nos *frames*, sem contudo haver transferência física dos dados.

Outra característica deste integrado é fornecer um esquema de prioridades para o tratamento de mensagens, com 4 filas de entrada e 4 de saída de mensagens, independentes entre si.

O processamento das mensagens tem requisitos de tempo real; logo, as mensagens disparam diversos temporizadores à medida em que vão sendo tratadas. Estes temporizadores são dedicados ao controle do fluxo de informação entre camadas, e à gerência de elementos ligados a este fluxo, por exemplo de "janelas" de protocolo, e ao controle de "timeouts" associados a cada entidade do protocolo.

### 6.2.2 Comunicação entre camadas

Adequando-se ao modelo OSI, mapeiam-se camadas do modelo em tarefas do núcleo de tempo real. A passagem de informação entre camadas é feita através de primitivas de serviço associadas às camadas; esta pode se dar, do ponto de vista da implementação, de duas formas:

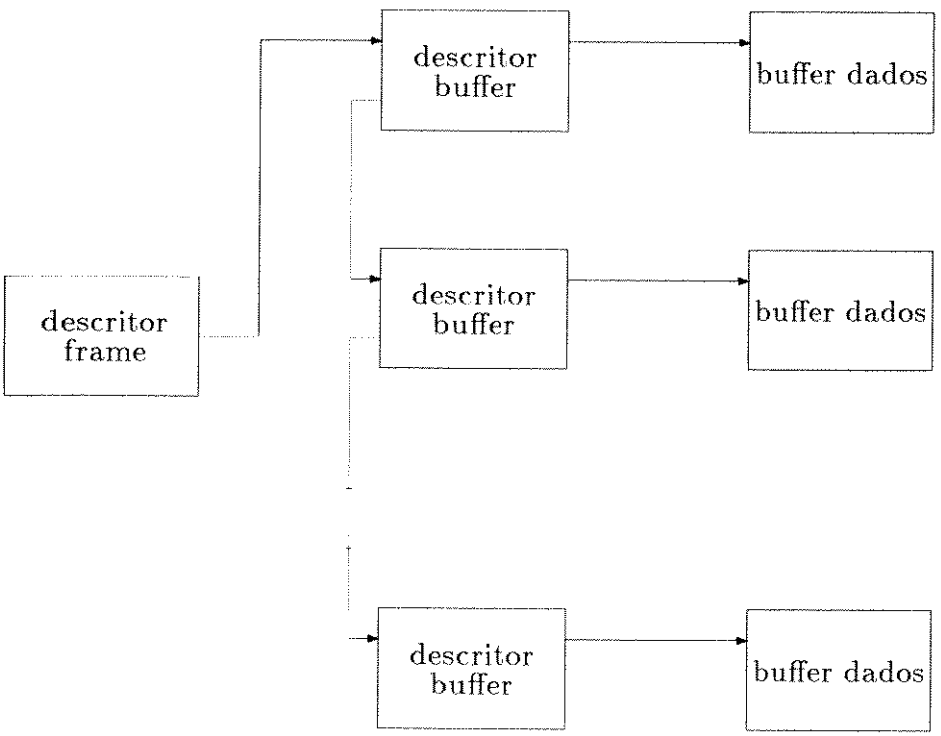



Figura 6.3: Estrutura de buffers no MC 68824

- definindo o formato como parâmetro de entrada para a tarefa que representa o protocolo
- definindo um bloco (ou buffer) para conter as informações necessárias à tarefa

Observa-se que o uso de um "buffer" uniformiza a forma de transferência de informação entre as tarefas, permitindo a definição de um formato único, preenchido com os parâmetros relevantes àquele protocolo ao qual se destina [60].

A comunicação entre camadas - protocolos - se dá através de "portas" (mecanismo comum de sincronização e comunicação entre tarefas, conforme descrito no capítulo 2). Trafegam nas portas endereços para os "buffers" de informação descritos acima; este mecanismo é usado de forma a diminuir a quantidade de cópias de dados entre as camadas.

### 6.2.3 Filosofias de execução das camadas

A necessidade de altas performances em sistemas de comunicação restringe as filosofias possíveis para seleção para posse do processador (escalonamento). Não são desejadas soluções genéricas, uma vez que as ações de uma entidade de protocolo são ordenadas e previsíveis.

O comportamento de um protocolo é geralmente definido a nível de máquina de estados. Este comportamento permite que o controle da máquina seja transferido de forma ordenada entre os elementos executantes do protocolo.

Esta característica possibilita que se definam filosofias de "scheduling" de tarefas adequadas à execução de protocolos, principalmente no tocante a tempo de execução.

## 6.3 Alternativas para implementação de serviços para protocolos

Do discutido no ítem anterior observa-se que o núcleo, para adequação à implementação de protocolos, deve fornecer mecanismos para processamento de mensagens e comunicação entre elementos do protocolo, e para o controle eficiente da posse do processador otimizados.

Conforme descrito no capítulo 2, no escopo de entidade essencial são fornecidos serviços para o controle do processador e a comunicação entre os elementos do protocolo. Estes serviços são configuráveis; portanto, a alteração da filosofia de escalonamento e a forma de passagem de parâmetros poderiam ser facilmente alteradas, desde que mantidas as interfaces a estes serviços conforme descrito no capítulo 3. Novas funções são previstas dentro da entidade de serviço à aplicação.

A seguir, apresenta-se uma discussão sobre como poderiam ser ampliadas e configuradas as funções de um núcleo para protocolos, dentro do modelo desenvolvido.

### 6.3.1 Gerência de buffers

O processamento de mensagens e a passagem de dados entre camadas sugere que exista um mecanismo de gerência de "buffers". Do ponto de vista do núcleo, é oferecido como entidade básica a gerência de memória, através da qual é controlado o fornecimento e a liberação de blocos da memória, tanto para a aplicação quanto para o próprio núcleo.

Para o caso de protocolos, a gerência de buffers opera a um nível superior àquele fornecido pelo esquema genérico de gerência de memória,

e englobaria como funções:

- a alocação prévia de um "buffer pool", através da gerência de memória
- o fornecimento à aplicação (protocolo) de "buffers" deste "pool", controlando a conexão a outros previamente fornecidos. Neste caso, supõe-se a nível do núcleo um controle de uma lista ligada de buffers, transparente ao usuário. Uma solução do tipo "PDU x quantidade de buffers necessários", ou "buffer para transferência de informação", poderia também ser adotada
- o controle da liberação de "buffers", coordenada em alto nível (ou seja, a nível de aplicação seria liberada, por exemplo, uma PDU e não uma quantidade de "buffers")
- a informação à aplicação da situação do "buffer pool"
- a alocação dinâmica de outros "pools" e a integração destes ao ambiente já existente

O uso do suporte fornecido por circuitos dedicados como o TCB *MC 68824* é facilmente integrado à gerência de buffers; de forma similar, seriam utilizados os serviços da gerência de memória, como entidade básica, configurando-se uma nova gerência, a nível de entidade de serviço à aplicação.

### 6.3.2 Gerência de temporizadores

Temporizadores são previstos a nível de entidade básica, utilizáveis portanto pelo próprio núcleo e pela aplicação. O número de temporizadores utilizados em implementação de protocolos sugere que operações com "timeouts" implícitos (como espera de mensagens em

portas) são adequadas, pois diminuem a quantidade eventual de temporizadores a serem manipulados.

A forma de implementação de temporizadores é variada, conforme discutido no capítulo 3. A alteração da forma de implementação, visando ajuste aos protocolos, enquadra-se no nível de configuração "small" do núcleo, conforme capítulo 4.

Uma forma possível de implementação desta gerência, para utilização em protocolos, é a criação de uma tarefa especial de temporização [60], comunicando-se com as demais tarefas através de portas. Desta forma, a tarefa "temporizador" se comporta como qualquer outra tarefa. Sua conexão à entidade essencial (serviços de gerência de interface com o hardware) pode ser mantida de forma a utilizar, como base de tempo, o relógio hardware fornecido pela máquina.

Porém, na implementação desta tarefa especial, a característica de temporização em portas pode ser usada como base de tempo. Neste caso, seriam criadas portas especiais, que não receberiam mensagens. A espera nestas portas geraria uma ocorrência de "timeout", e portanto, um evento para informação de tempo decorrido. Esta opção preferencia o uso de soluções já presentes no próprio núcleo (caso utilizada a implementação descrita no capítulo 5), evitanto incorporação de gerência adicionais, e portanto diminuindo código e desenvolvimento de novos algorítmos para controle de tempo.

### 6.3.3 Escalonamento

A necessidade do suporte fornecido por um núcleo *tempo real* à implementação de protocolos é patente considerando-se:

1. que as tarefas a serem executadas no tratamento da comunicação são independentes, e portanto podem ser executadas concorrentemente
2. que é necessária uma utilização efetiva do hardware de comunicação
3. que existem requisitos de tempo envolvidos tanto no próprio tratamento das mensagens quanto no de eventos de ocorrência assíncrona

No tocante a protocolos, um estudo aprofundado do comportamento da máquina de estados que descreve a operação do protocolo pode orientar a construção de um mecanismo de execução de tarefas adequado.

A função de escalonamento é classificada como entidade essencial no modelo desenvolvido. Conforme o capítulo 3, diversas filosofias podem ser usadas para sua implementação.

Um mecanismo de escalonamento entre tarefas, para protocolos, pode basear-se na predição do tempo de duração da tarefa - possível a partir dos estados da camada de protocolo que implementa. Desta maneira, a execução não-preemptiva passa a se tornar uma alternativa viável para a filosofia de escalonamento. Cumpre lembrar que as ações de uma entidade de protocolo não compõem um universo desordenado no qual interrupções entre quaisquer tarefas executantes devem ser previstas, e sim um sistema bastante hierarquizado. A cessão ordenada do controle entre tarefas, sem retorno ao escalonador enquanto as ações relacionadas a um evento não forem completamente executadas, é uma opção, exemplificada em [61].

Em [65], foram utilizadas soluções para escalonamento para obter alta performance. O "overhead" resultante do salvamento de contexto na

transferência do processador entre tarefas foi evitado através de um método de "ligação indireta". Neste método, existe uma rotina de interface entre as tarefas e o núcleo, que prepara convenientemente o ambiente ("contexto") para a transferência de execução de uma tarefa para outra sem intervenção do escalonador.

Outro tipo de solução é a opção "hard real time" [8], que assume a ocorrência de preempção somente no caso de vencimento de um tempo previamente associado à execução da tarefa.

É possível também a utilização de um esquema misto, com um conjunto de informações centralizado de forma a permitir ao controlador do escalonamento atuar conforme a situação das tarefas e das vias de comunicação (portas). A operação deste controlador ocorreria após a execução initerrupta da ação da tarefa, e a tomada de decisão sobre chaveamento de contexto seria baseada na situação do momento. Por exemplo, a alocação de mais uma "execução" a uma determinada tarefa, caso existam condições para tanto, "balanceando" a carga nas demais "camadas" (tarefas). Ou então a previsão de um tempo de execução menor para tarefas com maior quantidade de dados a processar, permitindo assim que tarefas com alta taxa de solicitações mas ações curtas fossem executadas com maior freqüência.

Um modelo de implementação é apresentado em [60]. Dados entre tarefas são passados através de "buffers" que, por sua vez, contêm informações sobre o estado presente da tarefa ao qual se destina e quais os tratamentos para os diversos estados que esta pode tratar. As tarefas são ativadas ao receberem estas informações (através dos buffers) via portas de comunicação inter-tarefas; porém, sua execução se dará de acordo com os dados de estado presentes no buffer. Com isso, obtém-se um escalonamento ordenado pelo fluxo de informações entre tarefas; todas ficam à espera em suas portas, e são escalona-

das quando da presença de uma mensagem em uma das portas. Do ponto de vista do núcleo, o escalonamento é feito entre camadas; é fornecida uma "máquina virtual" de execução a cada protocolo, e, com isso, pode-se obter diversos tratadores de um protocolo através da instanciação múltipla da tarefa que o implementa. As operações internas ao protocolo são parte de uma tarefa, como rotinas; como as execuções internas são seqüenciais, não há problema de "reentrância", pois, entre tarefas, existe uma troca efetiva de contexto e, intra-tarefa, cada etapa do tratamento (associada a um estado do protocolo) é "atômica".

Estas últimas propostas de solução incorporariam ao núcleo, no tocante ao escalonamento, uma certa "inteligência", permitindo que haja um ajuste mais fino entre as necessidades da aplicação e os serviços fornecidos.

## 6.4 Conclusões

A discussão sobre as características de implementação de protocolos mostra que um núcleo de tempo real que suporte tal aplicação pode ser especializado. Neste caso, funções antes implementadas pelo próprio protocolo são "transferidas" para o nível de software básico. Neste transferência mapearam-se os serviços nas entidades do modelo desenvolvido; observa-se que a alteração ou acréscimo das funções não força modificações drásticas no conjunto de código do núcleo, podendo ser feitas de forma independente.

A adequação simples de um núcleo a uma aplicação é o resultado principal deste trabalho, mostrando que o uso de uma filosofia estruturada para desenvolvimento e implementação do software básico permite um ajuste adequado às necessidades da aplicação. O exemplo

da análise para implementação de protocolos contribui para a prova desta adequação. A diversidade de políticas de implementação possibilita inclusive que vários núcleos diferentes sejam criados, até que se obtenha uma combinação ótima de gerências e entidades para a aplicação alvo.

# Apêndice A

# Exercício de implementação: pseudo-códigos

Serão descritos neste anexo os algorítmos que implementam cada função do núcleo implementado, em forma de pseudo-código, e separadas por entidades. Objetiva-se com isso fornecer uma documentação mais detalhada da implementação, de forma a possibilitar um conhecimento mais profundo do código e abrir maiores perspectivas de alterações/expansões.

Este exercício de implementação foi baseado na reutilização de diversos conjuntos de códigos, agrupados de acordo com o modelo desenvolvido, dentro de uma interface a serviços padrão conforme descrita no anexo B - pag. 180.

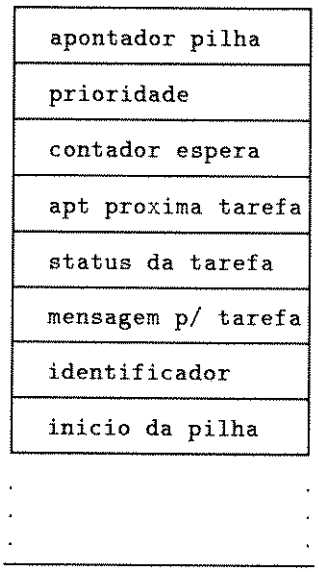
## A.1 Estruturas de dados

Será apresentado neste ítem o formato das principais estruturas de dados do núcleo. Elas são disponíveis à aplicação pelo arquivo "gl.h".

### A.1.1 TCB - bloco de controle de tarefa

A criação de uma tarefa para o núcleo compreende da alocação de um bloco de controle, que contém tanto informações sobre a tarefa como referência para o código que a implementa. É alocado também espaço para a pilha da tarefa. Uma vez criado fisicamente o TCB, a operação interna do núcleo sobre ele se dá através de apontadores. Para qualquer ação sobre a tarefa é no TCB que são fornecidas as informações essenciais. Por exemplo, o estado de execução (pronto ou suspenso), o tempo de execução, prioridade. A pilha da tarefa contém os valores dos registradores quando a tarefa foi criada, e posteriormente os do momento em que a ação da tarefa foi interrompida, possibilitando a reconstrução de seu contexto de execução.

O formato do TCB é mostrado na figura abaixo.



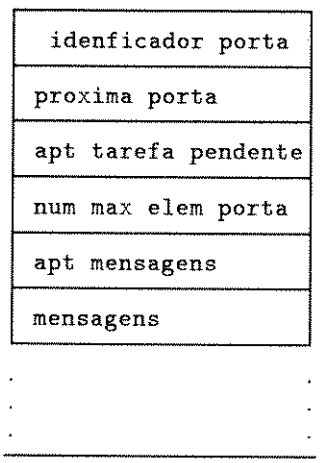

Este bloco tem tamanho variável, conforme o tamanho da pilha especificado para a tarefa. Um mínimo que possibilite a operação segura da tarefa é garantido no momento de sua criação.

### A.1.2 T_QUEUE - bloco descritor de filas

É um bloco de controle que existe para cada porta (que é, no sentido de implementação, uma fila). Nesta estrutura existem controles para manutenção de tarefas suspensas na porta à espera de mensagens e para manutenção de várias mensagens (na forma de seus apontadores). Esta estrutura é alocada no momento da criação de uma porta. Diversas estruturas representando portas são interligadas formando uma lista ligada, ordenada conforme a seqüência de criação. As tarefas eventualmente suspensas em uma porta são enfileiradas em uma lista ligada ordenada por ordem de chegada.

Existe ainda um valor específico que identifica a estrutura como descritor de porta, e outro que define o tipo de porta (entre normal e especial, que não recebe mensagens).

O formato de T_QUEUE é mostrado na figura abaixo.

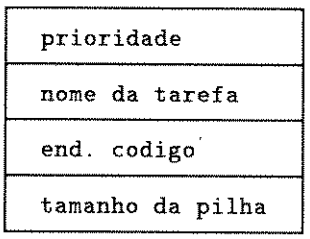



### A.1.3 T SEM - bloco controlador de semáforos

É idêntico ao formato de T_QUEUE.

### A.1.4 PCT - estrutura para passagem de parâmetros para criação de tarefas

Visando obter maior facilidade na passagem de parâmetros na criação de tarefas foi elaborada a estrutura PCT, onde o usuário define prioridade, nome, endereço o do código e tamanho da pilha associados à tarefa. O formato desta estrutura é mostrado na figura abaixo.




## A.2 Dados e definições do sistema

Nesta categoria incluem-se os dados globais utilizados no núcleo. São eles:

- a fila de processos à espera de ativação
- o ponteiro para o processo ativo no momento
- um contador de relógio
- um apontador para o início de uma lista ligada de portas
- um contador do número de tarefas criadas no núcleo

As definições necessárias são:

- a estrutura de controle de processos
- a estrutura de controle das portas
- os códigos de erro utilizados e eventuais macros de uso geral

## A.3 Algorítmo dos serviços da entidade essencial

*Observação*: neste e nos ítens subsequentes será usada a convenção para tipo de interface:

- A: a rotina é acessada tanto internamente ao núcleo como pela aplicação
- I: a rotina é acessada apenas internamente ao núcleo
- E: rotina de acesso externo (pela aplicação) exclusivo

e para a linguagem utilizada, A significa "assembly" e C, a linguagem C, segundo definição ANSI.

### A.3.1 Gerência de interface com o hardware

#### Programação do relógio da máquina

- nome: PROG_REL
- entrada: *fator* - fator de multiplicação do relógio base do PC
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: schedule.asm
- algorítmo:

```
faz interface padrao C-assembly
armazena valor do segmento de dados
armazena o fator de incremento do relogio
IF fator NEQ 0
  AND
   fator NEQ 1
  THEN
    programa novo valor de relogio
```

```
    FI
    obtem endereco de tratamento da interrupcao de re-
logio para rotina do nucleo
    armazena internamente
    reprograma tratamento da interrupcao de relogio para
rotina do nucleo
    retorna
```

### Restaura relógio da máquina

- nome: DESPROG_REL
- entrada: -
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: schedule.asm
- algorítmo:

```
    faz interface padrao C-assembly
    IF foi alterada a programacao do relogio
      THEN
        conecta tratamento original ao vetor de inter-
rupcao de relogio
    FI
    reprograma relogio como original
    retorna
```

### Serviço à interrupção de relógio

- nome: SERV
- entrada: (via interrupção de relógio)
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: schedule
- algorítmo:

```
    incrementa contador de numero de interrupcoes
    IF escalonador nao bloqueado
      THEN
        incrementa contador de entradas do escalonador
        salva conteudo dos registradores (contexto de
execucao)
        restaura segmento de dados original do inicio
da operacao
        armazena no TCB da tarefa ativa o apontador para
a pilha atual
        endereca pilha local para operacao do escalonador
        chama rotina de escalonamento (ESCALONA)
        zera contador de numero de interrupcoes
        restaura contexto de execucao para a nova tarefa
    FI
    decrementa contador de interrupcoes do relogio original
    IF numero de interrupcoes ocorridas EQ numero ori-
ginal de interrupcoes do relogio
```

```
        THEN
            reseta contador de interrupcoes do relogio original
            desvia tratamento para a rotina de servico ao
    relogio original
        ELSE
            libera o controlador de interrupcoes
            retorna
        FI
```

### Início da operação do escalonador

- nome: INI_ESC
- entrada: -
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: swap.asm
  obs: chamada no início da operação multitarefa
- algorítmo:

```
    salva contexto do inicio da operacao (referente a
aplicacao)
    altera teste de pilha original do compilador (ALTERA_CHK_PILHA)
    passa controle para a rotina de controle de "swap"
entre tarefas (SWAP_IN)
```

### Termina operação da tarefa

- nome: FIM_ESC
- entrada: errcode - código de erro na operação. É devolvido para o iniciador da operação multitarefa.
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: swap.asm
  obs: o controle retorna à rotina que iniciou a operação multitarefa
- algorítmo:

```
bloqueia interrupcoes
prepara valor de retorno com "errcode"
restaura pilha inicial (da aplicacao)
retira check de pilha (RESETA_CHK_PILHA)
desprograma o relogio (DESPROG_REL)
IF errcode EQ operacao ok
  THEN
    libera memoria relativa ao TCB da tarefa ativa
FI
retorna
```

### Instala tarefa no processador

- nome: INSTAL_TAR

- entrada: apontador para o TCB de uma tarefa
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: swap.asm
  obs: não há retorno desta rotina; o controle para diretamente à tarefa instalada. Deve ser chamada com o escalonador bloqueado.
- algorítmo:

```
    prepara pilha para a nova tarefa
    libera TCB da tarefa ativa
    passa controle para a rotina de "swap" entre tare-
fas (SWAP_IN)
```

**Executa "swap" entre tarefas**

- nome: SWAP_IN
- entrada: apontador para TCB de uma tarefa
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: swap.asm
  obs: a operação do escalonador deve estar bloqueada durante a execução;
  entrega à tarefa representada na entrada o processador;
  libera o escalonador após efetuar a troca de contexto.

- algorítmo:

```
    bloqueia interrupcoes
    salva contexto corrente
    salva no TCB da tarefa ativa apontador para a pi-
lha atual
    coloca a tarefa referenciada na entrada como tarefa
corrente
    obtem nova pilha no TCB da nova tarefa corrente
    restaura contexto (registradores)
    libera operacao do escalonador (RELEASE)
    libera interrupcoes
    retorna
```

### Altera check de pilha

- nome: ALTERA_CHK_PILHA
- entrada: -
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: swap.asm
- algorítmo:

```
    obtem endereco da rotina original do compilador para
teste de pilha
    armazena conteudo das primeiras instrucoes desta rotina
```

```
    escreve sobre estes enderecos despachador para ro-
tina interna de teste
    habilita o teste de pilha interno as tarefas
    retorna
```

**Restaura check original de pilha**

- nome: RESETA_CHK_PILHA
- entrada: -
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: swap.asm
- algorítmo:

```
    restaura codigo original da rotina de teste de pi-
lha do compilador
    retorna
```

**teste local de pilha**

- nome: MYCHKTSK
- entrada: número de bytes necessários aos dados locais
- saída: -
- linguagem: A

- tipo de interface: I
- módulo: swap.asm
- algorítmo:

```
IF check de pilha habilitado
  THEN
    IF apontador do topo de pilha LE tamanho do TCB
      THEN
        indica erro grave
        termina operacao do escalonador (FIM_ESC)
        /* nao deveria retornar, mas por protecao...*/
        retorno forcado ao DOS
    FI
FI
abre espaco na pilha para dados locais
retorna
```

### Bloqueia operação do escalonador

- nome: BLOCK
- entrada: -
- saída: s
- linguagem: A
- tipo de interface: I
- módulo: schedule.asm
- algorítmo:

```
coloca NAO_ESCALONA na variavel de controle de escalonamento
retorna
```

**Libera operação do escalonador**

- nome: RELEASE
- entrada: -
- saída: -
- linguagem: A
- tipo de interface: I
- módulo: schedule.asm
- algorítmo:

```
coloca ESCALONA na variavel de controle de escalonamento
retorna
```

**Aloca memória**

- nome: DMALLOC
- entrada: quantidade de bytes a serem alocados
  endereço para receber apontador para o bloco a ser alocado
- saída: OPERACAO_OK, NO_MEM
- linguagem: C
- tipo de interface: A

- módulo: dmalloc.c
- algorítmo:

```
aloca memoria via "malloc"
IF nao tem sucesso
  THEN
    retorna NO_MEM
FI
coloca em tabela interna de controle (ADD_TO_TABLE)
IF nao sucesso
  THEN
    retorna NO_MEM
FI
retorna OPERACAO_OK
```

### Libera memória

- nome: DFREE
- entrada: apontador para o bloco de memória a ser liberado
- saía: PAR_INCORRETO, OPERACAO_OK
- linguagem: C
- tipo de interface: A
- módulo: dmalloc.c
- algorítmo:

```
verifica existencia do apontador para o bloco
```

```
   referenciado em tabela interna (REMOVE_FROM_TABLE)
IF nao existe o apontador
  THEN
    retorna PAR_INCORRETO
ELSE
    libera memoria com "free"
    retorna OPERACAO_OK
FI
```

### Busca apontador de bloco de memória em tabela

- nome: REMOVE_FROM_TABLE
- entrada: apontador a ser verificado
- saída: 0 se não achou
  1 se achou
- linguagem: C
- tipo de interface: I
- módulo: dmalloc.c
- algorítmo:

```
busca apontador em tabela interna (FIND_IN_TABLE)
IF achou
  THEN
    retira apontador da tabela
    retorna 1
ELSE
    retorna 0
```

```
FI
```

### Coloca apontador em tabela

- nome: ADD_TO_TABLE
- entrada: apontador a ser colocado na tabela
- saída: 0 se não colocou o apontador na tabela
  1 se colocou (operação com sucesso)
- linguagem: C
- tipo de interface: I
- módulo: dmalloc.c
- algorítmo:

```
busca apontador na tabela interna (FIND_IN_TABLE)
IF achou
  THEN
    retorna 0 (operacao incorreta)
ELSE
    coloca apontador na tabela interna
    retorna 1 (operacao com sucesso)
FI
```

### Operação sobre tabela interna de apontadores para blocos de memória

- nome: FIND_IN_TABLE

- entrada: ordem do ítem buscado
  endereço para resposta
- saída: 0 se não encontrado, ou a ordem do ítem
- linguagem: C
- tipo de interface: I
- módulo: dmalloc.c
- algorítmo:

```
IF nao ha' nada na tabela
  THEN
    retorna apontador para a tabela
FI
WHILE ha' elementos na tabela
    faz busca binaria
ENDWHILE
retorna posicao onde foi encontrado o elemento
```

### A.3.2 Gerência de escalação de tarefas

#### Criação de tarefass

- nome: CRIA_TAREFA
- entrada: par - parâmetros para a tarefa segundo a estrutura PCT
  endereço para devolver o apontador para o TCB da tarefa criada
  parâmetros para a tarefa a ser criada

- saída: número máximo de tarefas já existe no núcleo
  não há memória para a operação
  operação ok
- linguagem: C
- tipo de interface: A
- módulo: proc.c
- algorítmo:

```
    IF numero maximo de tarefas excedido
      THEN
        retorna NMAX_PROC
    FI
    prepara parametros para criacao da tarefa
    obtem parametros que passarao para a tarefa
    bloqueia operacao do escalonador (BLOCK)
    IF tamanho da pilha para a nova tarefa LE 512
      THEN
        tamanho de pilha para a nova tarefa = 512
    FI
    aloca memoria para TCB da tarefa a ser criada e sua
pilha
    IF nao ha' memoria para a operacao
      THEN
        retorna NO_MEM
    FI
    IF nao existe fila de TCBs (primeira tarefa criada)
      THEN
        cria fila de TCBs (CRIA_FILA)
```

```
    FI
    inicia TCB com zero
    inicia pilha com padrao a5a5
    inicia campos da TCB segundo parametros da criacao
da tarefa
    inicia pilha da tarefa com os registradores em seu
valor atual
     e parametros que passarao para a tarefa
    IF prioridade da tarefa criada GT prioridade da ta-
refa ativa
      THEN
        insere tarefa ativa na fila de TCBs prontos (INS_FILA)
        transfere controle para a tarefa criada (SWAP_IN)
    ELSE
        insere tarefa criada na fila de TCBs prontos (INS_FILA)
    FI
    libera operacao do escalonador (RELEASE)
    retorna OPERACAO_OK
```

### Destruição de tarefas

- nome: DESTROI TAREFA
- entrada: apontador para TCB da tarefa a ser destruida
- saída: tarefa não existe, deadlock, operação ok
- linguagem: C
- tipo de interface: A
- módulo: proc.c
- algorítmo:

```
    bloqueia escalonador (BLOCK)
    IF tarefa a ser deletada EQ tarefa ativa
      THEN
        IF nao ha' outras tarefas prontas
          THEN
            termina operacao multitarefa com DEADLOCK
(FIM_ESC)
        FI
        IF ha' outras tarefa mas estao suspensas
          THEN
            termina operacao multitarefa com OPERACAO_OK
(FIM_ESC)
        FI
        retira tarefa da fila dos TCBs prontos
        instala a tarefa retirada no processador (INSTAL_TAR)
        decrementa o numero de tarefas presentes no nucleo
        libera memoria alocada a TCB ativa e sua pilha
    ELSE
        busca tarefa referenciada na fila de prontos (REMOVE_FILA)
        IF encontra tarefa na fila de prontos
          THEN
            retira a tarefa da fila (DEL_FILA)
            libera memoria alocada a TCB e sua pilha
        ELSE
            busca tarefa nas portas (BUSCA_PROC)
            IF encontra tarefa
              THEN
                retira tarefa da fila (DEL_QUEUE)
                libera memoria alocada a TCB e sua pilha
```

```
            ELSE
                libera escalonador (RELEASE)
                retorna TAREFA_NAO_EXISTE
            FI
        FI
    FI
    libera escalonador (RELEASE)
    retorna OPERACAO_OK
```

**Reescalona**

- nome: REESCALONA
- entrada: -
- saída: OPERACAO_OK
- linguagem: C
- tipo de interface: A
- módulo: proc.c
- algorítmo:

```
bloqueia escalonador (BLOCK)
IF ha' tarefas a escalonar
  THEN
    retira tarefa da fila de TCBs prontos (DEL_FILA)
    insere tarefa ativa na fila de TCBs prontos (INS_FILA)
    transfere controle para a nova tarefa ativa (SWAP_IN)
ELSE
    libera escalonador (RELEASE)
```

```
        retorna OPERACAO_OK
    FI
```

**Suspende tarefa**

- nome: SUSPENDE_TAREFA
- entrada: quantidade de tempo máximo (em segundos) que a tarefa pode ficar suspensa
- saída: deadlock, operação ok
- linguagem: C
- tipo de interface: E
- módulo: proc.c
- algorítmo:

```
IF e' a unica tarefa ativa do nucleo
  THEN
    retorna DEADLOCK
FI
coloca tarefa na fila de suspensos (ESPERA_MENS)
 com opcoes de consumo, por tempo maximo especificado
retorna OPERACAO_OK
```

**Retira tarefa da fila de suspensos**

- nome: RESUME_TAREFA
- entrada: identificador da tarefa (apontador para TCB) a ser retirada da suspensão

- saída: tarefa não existente, operação ok
- linguagem: C
- tipo de interface: E
- módulo: proc.c
- algorítmo:

```
bloqueia escalonador (BLOCK)
busca tarefa na fila de suspensos (BUSCA_PROC)
IF encontra tarefa
  THEN
```

### Altera prioridade de tarefa

- nome: MUDA_PRI
- entrada: identificador da tarefa (apontador para TCB) a ter a prioridade alterada
  a nova prioridade do elemento
- saída: parâmetro incorreto, tarefa não existente, operação ok
- linguagem: C
- tipo de interface: E
- módulo: proc.c
- algorítmo:

```
    IF nova prioridade maior que a maxima prioridade de
tarefas no nucleo
     THEN
```

```
      retorna PAR_INCORRETO
    FI

    bloqueia escalonador (BLOCK)
    IF tarefa a ter prioridade alterada e' a tarefa ativa
     THEN
      altera prioridade da tarefa ativa
      provoca escalonamento (REESCALONA)
    ELSE
      IF a tarefa referenciada esta' na fila de prontos
       THEN
        retira da fila (REM_FILA)
altera prioridade
coloca na fila (INS_FILA)
reescalona (REESCALONA)
      FI
      IF tarefa referenciada esta' suspensa em porta
       THEN
        retira da fila (DEL_QUEUE)
altera prioridade
coloca na fila de prontos (INS_FILA)
reescalona (REESCALONA)
       ELSE
        libera escalonador (RELEASE)
retorna TAREFA_NAO_EXISTE
      FI

    retorna OPERACAO_OK
    FI
```

**Busca tarefa em fila**

- nome: BUSCA_PROC
- entrada: tarefa a ser buscada (apontador para seu TCB)
  fila onde deve ser buscada a tarefa
  tipo de busca (verificação ou consumo)
- saída: NULL se não encontrou a tarefa, ou o próprio TCB da tarefa encontrada.
- linguagem: C
- tipo de interface: I
- módulo: proc.c
- algorítmo:

```
    FOR inicio da fila de tarefas suspensas TO fim da
fila de tarefas suspensas
     DO
      IF tarefa suspensa OR tarefa buscada encontrada
       THEN
        IF tipo de busca = CONSOME
 THEN
  retira da fila
FI
retorna identificador da tarefa
      FI
    ENDFOR
```

```
retorna NULL
```

### Retira tarefa suspensa em porta

- nome: DEL_QUEUE
- entrada: identificador da tarefa procurada (apontador para TCB)
- saída: NULL se não encontrou a tarefa ou o próprio apontador para o TCB encontrado
- linguagem: C
- tipo de interface: I
- módulo: proc.c
- algorítmo:

```
    FOR inicio das portas TO fim das portas
     DO
      FOR todos os TCB's suspensos a espera de mensagem
       DO
        IF encontrou o TCB referenciado
         THEN
  retira da fila
  retorna o apontador para o TCB
FI
      ENDFOR
    ENDFOR
```

**Escalonador**

- nome: ESCALONA
- entrada: -
- saída: -
- linguagem: C
- tipo de interface: I
- módulo: proc.c
- algorítmo:

```
    suspende testes sobre pilha de execucao
    incrementa contador de execucao do escalonador (estatistica)
    atualiza contador de relogio com o valor do conta-
dor de
      interrupcoes sem tratamento pelo escalonador
    reseta este contador
    busca tarefas com timeout na espera em filas
    IF tarefa com timeout
     THEN
      coloca tarefa na fila de prontos (INS_FILA)
    FI
    incrementa contador de tempo da tarefa ativa
    verifica fila de prontos
    IF prioridade da tarefa ativa menor que
       a prioridade maior de uma tarefa pronta
     THEN
       troca tarefa pronta com tarefa ativa
    FI
```

```
retorna testes sobre a pilha
retorna
```

### Comparação entre dois elementos da fila de prontos

- nome: pq_cmp
- entrada: apontadores para TCB's a serem comparadas (A e B)
- saída: ¿ 0 se prioridade de A ¿ B
  ¡ 0 se prioridade de A ¡ B
- linguagem: C
- tipo de interface: I
- módulo: proc.c
  OBS: usada como parâmetros para INS_FILA
- algorítmo:

```
IF prioridade de A difere da prioridade de B
 THEN
  retorna prioridade A - prioridade B
ELSE
  retorna tempo de existencia A - tempo de existen-
cia B
FI
```

### Troca entre elementos da fila

- nome: pq_swap

- entrada: apontadores para TCB's a serem trocadas
- saída: -
- linguagem: C
- tipo de interface: I
- módulo: proc.c OBS: usada como parâmetro para "INS_FILA"
- algorítmo:

```
troca apontadores entre si
```

**Inicia operação multitarefa**

* nome: INICIA OPERAÇÃO
* entrada: fator de tempo que incrementa o relógio da máquina
* saída: -
* linguagem: C
* tipo de interface: E
* módulo: gint.c
* algorítmo:

```
cria porta para suspensao de tarefas (CRIA_PORTA)
cria tarefa idle para evitar deadlock (CRIA_TAREFA)
IF fator de tempo > 0
 THEN
  programa novo fator de relogio
FI
mapeia ^C para terminar operacao multitarefa
inicia operacao do escalonador (INI_ESC)
```

OBS: a rotina não retorna, uma vez que o controle passa às tarefas presentes no núcleo.

### A.3.3 Gerência de sincronização e comunicação

#### Cria porta no núcleo

* nome: CRIA PORTA
* entrada: número de mensagens que a porta pode conter
  tipo da porta
  endereço para devolver o apontador referente a porta criada
* saída: parâmetro incorreto
  não existência de memória
  operação ok
* linguagem: C
* tipo de interface: A
* módulo: porta.c
* algorítmo:

```
IF tipo da porta diferente de normal ou especial
 THEN
  retorna PAR_INCORRETO
FI

IF numero de mensagens especificado para a porta
maior
    que o numero maximo de mensagens esperado no
nucleo
```

```
     THEN
      retorna PAR_INCORRETO
    FI

    bloqueia operacao do escalonador (BLOCK)
    aloca memoria para controle da porta (T_QUEUE)
mais espaco
      para apontadores de  mensagens que a porta vai
conter
    libera escalonador (RELEASE)

    IF nao consegue alocar memoria
     THEN
      retorna NO_MEM
    FI

    inicia campos de controle da porta
    bloqueia escalonador (BLOCK)

    IF e' a primeira porta criada
     THEN
      cria fila de portas iniciando pela criada no
momento
    ELSE
      coloca porta criada na fila em forma de lista
ligada
    FI

    libera escalonador (RELEASE)
    coloca apontador para a porta no lugar para resposta
```

```
retorna OPERACAO_OK
```

### Libera porta

* nome: LIBERA_PORTA
* entrada: identificador da porta a ser liberada
* saída: parâmetro incorreto
  porta ocupada
  operação ok
* linguagem: C
* tipo de interface: A
* módulo: porta.c
* algorítmo:

```
IF nao e' identificador de porta
 THEN
  retorna PAR_INCORRETO
FI

IF nao ha' tarefas suspensas a espera na porta
E
   nao ha' mensagens na porta
 THEN
   libera memoria alocada a porta
   retorna OPERACAO_OK
ELSE
   retorna PORTA_OCUPADA
FI
```

**Espera mensagem em porta**

* nome: ESPERA_MENS
* entrada: identificador da porta
  tempo máximo de espera (segundos) na porta
  tipo de verificação (consumo ou verificação de mensagem)
  apontador para a mensagem a ser consumida
* saída: parâmetro incorreto
  timeout
  deadlock
  operação ok
* linguagem: C
* tipo de interface: A
* módulo: porta.c
* algorítmo:

```
IF nao e' um identificador de porta
 THEN
  retorna PAR_INCORRETO
FI

IF tipo de verificacao nao valido
 THEN
  retorna PAR_INCORRETO
FI

bloqueia escalonador (BLOCK)
IF ha' mensagem na porta
 THEN
  IF espera com consumo
```

```
        THEN
          consome mensagem (CONSOME_MENS)
        ELSE
          obtem numero de mensagens na porta
        FI
      ELSE
        IF nao ha' tarefas prontas
         THEN
          retorna DEADLOCK
        ELSE
          prepara tarefa ativa para ser suspensa
coloca tarefa ativa em suspensao na porta
obtem tarefa pronta para ser ativada (DEL_FILA)
passa controle para nova tarefa ativa (SWAP_IN)

/* .... ponto de retorno apos mensagem enviada a porta
ou timeout ....*/

          IF obteve mensagem
 THEN
  entrega mensagem ao solicitante
  retorna OPERACAO_OK
ELSE
  retorna TIMEOUT
FI
        FI

    libera escalonador (RELEASE)
    retorna OPERACAO_OK
```

### Consome mensagem

* nome: CONSOME_MENS
* entrada: identificador da porta
* saída: apontador para mensagem
* linguagem: C
* tipo de interface: I
* módulo: porta.c
* algorítmo:

```
decrementa numero de mensagens da porta
atualiza apontadores de mensagens da porta
entrega mensagem a tarefa ativa
retorna apontador para a mensagem
```

### Obtém mensagem na porta

* nome: OBTÉM_MENSAGEM
* entrada: identificador da porta
  endereço para entregar a mensagem obtida
* saída: parâmetro incorreto
  não existência de mensagens
  operação ok
* linguagem: C
* tipo de interface: E
* módulo: porta.c
* algorítmo:

```
IF porta referenciada nao valida
 THEN
  retorna PAR_INCORRETO
FI

IF ha' mensagem na porta
 THEN
  bloqueia escalonador (BLOCK)
  consome mensagem (CONSOME_MENS)
  libera escalonador (RELEASE)
  retorna OPERACAO_OK
ELSE
  retorna NO_MENS
FI
```

**Envia mensagem para porta**

* nome: ENVIA_MENS
* entrada: identificador da porta
  apontador para a mensagem
* saída: parâmetro incorreto
  porta ocupada
  operação ok
* linguagem: C
* tipo de interface: A
* módulo: porta.c
* algorítmo:

```
IF porta referenciada nao e' valida
 THEN
  retorna PAR_INCORRETO
FI

IF porta e' especial
 THEN
  retorna PAR_INCORRETO
FI

bloqueia escalonador (BLOCK)
IF porta esta' cheia
 THEN
  retorna PORTA_OCUPADA
FI

coloca mensagem na porta
IF ha' tarefa a espera
 THEN
  retira tarefa da espera
  retira mensagem da porta
  entrega mensagem a tarefa
  coloca tarefa na fila de prontos (INS_FILA)
  reescalona (REESCALONA)
FI

libera escalonador (RELEASE)
retorna OPERACAO_OK
```

## A.4 Algorítmos dos serviços da entidade básica

Nesta seção serão descritas as implementações ligadas às gerências do núcleo desenvolvido como complementação deste trabalho. A diversidade de serviços que podem fazer parte desta entidade faz necessário que se restrinja a discussão exclusivamente àqueles que foram realmente utilizados, ou seja, as gerências relacionadas a filas e memória. A gerência de E/S vai depender mais dos recursos disponíveis no hardware da máquina, como já foi discutido no capítulo 3. Outros serviços, como a gerência de calendário, não foram julgados necessários à presente implementação.

### A.4.1 Gerência de filas

Como discutido no capítulo 3, são várias as formas com as quais se pode implementar um mecanismo de gerência de filas. Será descrito aqui o código referente à implementação de uma fila que está sempre ordenada segundo uma chave, à qual se referirá doravante como "prioridade". Esta chave pode ser um valor qualquer, descrito no cabeçalho da fila, que no caso tem o formato "T_QUEUE". Outros campos podem ser acrescentados à esta estrutura, mantendo-se entretanto os elementos básicos de controle.

Segue a descrição dos serviços desta gerência

#### Rearruma fila de cima para baixo

* nome: REHEAP_DOWN

```
* entrada: descritor da fila
  apontador para o início da fila
* saída: -
* linguagem: C
* tipo de interface: I
* módulo: pq.c
* algorítmo:

     FOR primeiro elemento TO ultimo elemento da fila
      DO
       aponta para um lado da arvore representada pela
fila
       IF existem elementos neste lado
        THEN
         busca elemento a distancia 1
IF elemento a frente > elemento atual
 THEN
  troca elementos de lugar
FI
       FI

       IF topo da arvore > nivel inferior
        THEN
         termina a arrumacao
       ELSE
         troca topo com nivel inferior
desce um nivel na arvore
       FI
    ENDFOR
```

### Rearruma fila de baixo para cima

* nome: REHEAP_UP
* entrada: descritor da fila
* saída: -
* linguagem: C
* tipo de interface: I
* módulo: pq.c
* algorítmo:

```
aponta para fim da fila
WHILE e' possivel dividir a fila ao meio
 DO
  divide a fila
  IF elemento do fim > elemento do meio
   THEN
    termina a arrumacao
  FI

  troca elementos de lugar
  sobe um nivel na arvore
ENDWHILE
```

### Cria fila

* nome: CRIA_FILA

* entrada: número de elementos da fila
  tamanho de um elemento da fila
  apontador para rotina de comparação entre elementos da fila
  apontador para rotina de troca entre elementos da fila
  fila já ordenada para ser incorporada à que vai ser criada (NULL se não definida)
* saída: apontador para o identificador da fila
* linguagem: C
* tipo de interface: A
* módulo: pq.c
  OBS: a fila é ordenada segundo o algorítmo definido na rotina de comparação. Para tanto, INSERIR CAP. 3, PAG. 11.
* algorítmo:

```
calcula tamanho da fila em bytes
IF  ha' fila a ser agregada
 THEN
  aloca memoria apenas para controle da fila
  IF ocorre erro na alocacao de memoria
   THEN
    retorna 0
  FI
  coloca fila dentro do controle criado
  inicia apontador para posicao de armazenamento
  inicia campos de controle especificos da
      criacao com fila agregada
ELSE
```

```
        aloca memoria para fila (controle + armazenamento)
        IF ocorre erro na alocacao de memoria
         THEN
          retorna 0
        FI
        inicia apontador para area de armazenamento
        inicia campos de controle especificos da nova
criacao
    FI

    inicia campos de controle comum (apontadores para
rotinas
          de troca e comparacao, tamanho de elementos,
          numero maximo de elementos na fila)

    retorna apontador para fila
```

**Insere elemento na fila**

* nome: INS_FILA
* entrada: apontador para descritor da fila
  apontador para o elemento a ser inserido
* saída: espaço para inserção antes da inclusão
  (0 se a fila está cheia e não houve inserção)
* linguagem: C
* tipo de interface: A
* módulo: pq.
* algorítmo:

```
IF ha' espaco na fila
 THEN
  incrementa contador de elementos na fila
  copia elemento para a fila
  reordena de baixo para cima (REHEAP_UP)
FI
retorna espaco na fila antes da insercao
```

### Deleta elementos da fila

* nome: DEL_FILA
* entrada: apontador para o descritor da fila
  endereço para colocar o ítem retirado
* saída: número de ítens na fila antes da retirada
  (0 se a fila estava vazia)
* linguagem: C
* tipo de interface: A
* módulo: pq.c
* algorítmo:

```
IF ha' elementos na fila
 THEN
  obtem apontador para o primeiro item da fila
  copia o elemento para o lugar referenciado pela
entrada
  copia o ultimo elemento da fila para a posi-
cao liberada
```

```
        diminui de 1 o numero de elementos da fila
        reordena de cima para baixo (REHEAP_DOWN)
    FI

    retorna o numero de elementos na fila antes da
retirada
```

### Troca elemento mais prioritário com outro referenciado

* nome: TROCA FILA
* entrada: apontador para identificador da fila
  elemento a ser "trocado" com o primeiro da fila
* saída: número de posições usadas na fila
* linguagem: C
* tipo de interface: A
* módulo: pq.c
* algorítmo:

```
    troca primeiro elemento da fila com o
       referenciado na entrada da rotina
    reordena de baixo para cima (REHEAP_DOWN)
    retorna o numero de posicoes usadas na fila
```

### Remove um elemento arbitrário da fila

* nome: REMOVE FILA
* entrada: apontador para identificador da fila
  endereço para receber o elemento a ser retirado

endereço de uma rotina para localização do elemento na fila
item a ser comparado para localizar o elemento na fila

* saída: 0 se a fila está vazia
número de posições usadas na fila antes da retirada do elemento
* linguagem: C
* tipo de interface: A
* módulo: pq.c
* algorítmo:

```
FOR primeiro elemento da fila TO ultimo elemento
 DO
  busca elemento
  IF achou elemento
   THEN
    termina FOR
  FI
ENDFOR

IF achou elemento
 THEN
  copia para o endereco referenciado na entrada
  diminui de 1 o numero de itens da fila
  reordena a fila (REHEAP_DOWN)
FI
```

### Retorna apontador para início da fila

* nome: LOOK

* entrada: identificador da fila
* saída: elemento no topo da fila
  NULL se a fila estiver vazia
* linguagem: C
* tipo de interface: A
* módulo: pq.c
* algorítmo:

```
IF ha' elementos na fila
 THEN
  retorna apontador para o primeiro elemento da
fila
ELSE
  retorna NULL
FI
```

### Retorna número de elementos na fila

* nome: NUMELE
* entrada: identificador da fila
* saída: número de elementos na fila
* linguagem: C
* tipo de interface: A
* módulo: pq.c
* algorítmo:

```
retorna numero de elementos na fila
```

### A.4.2 Gerência de memória

**Fornece blocos de memória**

* nome: PEDE_MEM
* entrada: quantidade de bytes a serem alocados
  endereço para receber apontador do bloco alocado
* saída: OPERACAO_OK, NO_MEM
* linguagem: C
* tipo de interface: E
* módulo: mem.c
* algorítmo:

```
aloca bloco (DMALLOC)
retorna
```

**Libera blocos de memória**

* nome: LIBERA_MEM
* entrada: apontador para o bloco a ser liberado
* saída: PAR_INCORRETO, OPERACAO_OK
* linguagem: C
* tipo de interface: E
* módulo: mem.c
* algorítmo:

```
libera bloco (DFREE)
retorna
```

**Informa memória disponível**

* nome: MEM_DISP
* entrada: endereço para receber a quantidade de memória disponível
* saída: -
* linguagem: C
* tipo de interface: E
* módulo: mem.c
* algorítmo:

```
solicita memoria usada (MEMORY_USED)
retorna
```

# Apêndice B

# Interfaces funcionais para o núcleo implementado

Neste item serão descritas as interfaces aos serviços fornecidos pelas gerências anteriormente delineadas. Estas interfaces aqui descritas são complementadas por uma listagem dos "prototypes" (formas de chamada em "C", com especificações dos tipos dos parâmetros e valores de retorno), mais específica da implementação

## B.1 Gerência de Interrupções

### B.1.1 Bloqueia_interrupções

* **função:**
  Bloqueia a ocorrência de interrupções na máquina
* **formato de chamada:**
  bloqueia_interrupção();
* **parâmetros de entrada:**
  -
* **parâmetros de saída:**
  -
* **descrição:**
  Bloqueia a ocorrência de interrupções na máquina
* **erros possíveis:**
  -

### B.1.2 Libera_interrupção

* **função:**
  Habilita a ocorrência de interrupções na máquina
* **formato de chamada:**

libera_interrupção();

* **parâmetros de entrada:**

  -

* **parâmetros de saída:**

  -

* **descrição:**

  Libera a ocorrência de interrupção na máquina.

* **erros possíveis:**

  -

### B.1.3 Bloqueia_escalonador

* **função:**

  Bloqueia a função do escalonador

* **formato de chamada:**

  bloqueia_escalonador();

* **parâmetros de entrada:**

  -

* **parâmetros de saída:**

  -

* **descrição:**

  Bloqueia a função do escalonador no sentido de garantir a posse do processador pela tarefa ativa, não permitindo a troca de contexto mesmo na ocorrência de condição para tal. As interrupções continuam ativas na máquina.

* **erros possíveis:**

  -

### B.1.4 Libera escalonador

* **função:**
  Permite o funcionamento normal do escalonador, após a chamada de "bloqueia_escalonador".
* **formato de chamada:**
  libera_escalonador();
* **parâmetros de entrada:**
  -
* **parâmetros de saída:**
  -
* **descrição:**
  A função permite o funcionamento do escalonador de forma a que se dêem as trocas de contexto pertinentes. Não provoca reescalonamento.
* **erros possíveis:**
  -

### B.1.5 Inicia_operação

[1]

* **função:**
  Inicia a operação multitarefa do núcleo.
* **formato de chamada:**
  inicia_operação(fator_tempo);
* **parâmetros de entrada:**
  fator_tempo: fator pelo qual a operação do relógio da máquina será alterado.

[1] esta função é bastante particular para a implementação atual; pode servir como guia para outras implementações

* **parâmetros de saída:**

  -

* **descrição:**

  A rotina inicia a operação multitarefa do núcleo. O fator de tempo passado como parâmetro altera a freqüência do relógio da máquina. Se 0, implica em que o sistema não terá preempção - ou seja, as tarefas terão que explicitamente liberar o processador para que outras venham a executar suas funções. Um fator de tempo 1 conserva a taxa normal do relógio - 18,2 interrupções/segundo. Se o fator for muito grande o sistema pode se tornar mais lento porque vai perder interrupções. É aconselhável que o fator tempo seja uma potência de 2. Isso permite que se mantenha mais facilmente a função de relógio original da máquina. Após reprogramação do relógio o controle passa para a tarefa mais prioritária do núcleo.

  Esta função trata também da criação de uma tarefa com a menor prioridade possível, que impede erros devidos à não presença de tarefas no sistema ("deadlocks"). Sua identificação é fixa ("T0"), e não deve ser deletada pela aplicação. Esta tarefa cria a fila de controle de suspensão de tarefas. Quando for a única no sistema, esta tarefa deleta a fila de suspensos,e se destrói, terminando a operação multitarefa.

  Ao fim da operação multitarefa o controle retorna ao ponto logo após a chamada a esta função. Neste caso pode-se chamar de "valores de retorno" os códigos devolvidos após a operação toda do núcleo, como falta de memória, deadlock, estouro de pilha, operação correta, entre outros. No caso de retorno como "OPERAÇÀO OK" toda a memória utilizada

na operação foi liberada. Em retorno irregular pode ter havido erros na liberação da memória.

* **erros possíveis:**

  O caso onde o fator de tempo é incorreto (valor menor que 0) é ignorado e o relógio do sistema não sofre alteração.

## B.2 Gerência de processos

### B.2.1 Cria tarefa

* **função:**

  Faz a criação de uma tarefa para o núcleo.

* **formato de chamada:**

  s = cria_tarefa(endereço_parâmetros,&id_tarefa, ...);

* **parâmetros de entrada:**

  - endereço parâmetros: apontador para uma área que contém as características do processo, do tipo PCT (ver anexo A - pag. 130):

* **parâmetros de saída:**

  - NMAX PROC
  - NO_MEM;
  - OPERAÇÃO OK, e "id tarefa" contém a identificação da tarefa criada.

* **descrição:**

  Faz a criação de uma tarefa para o núcleo através da alocação de uma estrutura de dados (ver anexo A - pag. 130) que descreve o ambiente no qual a tarefa é executada e contém informações gerais sobre a mesma, obtidas a partir

dos parâmetros de entrada. O tamanho da pilha é assegurado pela função em um mínimo para a operação correta da tarefa. A tarefa, agora sob a forma de seu descritor, tem condições para entrar na operação do núcleo. A identificação da tarefa criada é passada, no caso de operação correta, no parâmetro "id_tarefa" de saída. Há um reescalonamento se a tarefa criada for de maior prioridade que a criadora.

* **erros possíveis:**
  Falta de recursos (não existência de espaço para alocação de um novo descritor de tarefas). Número máximo de processos no núcleo já existe.

### B.2.2 Destrói_tarefa

* **função:**
  Termina uma tarefa no núcleo.
* **formato de chamada:**
  s = destrói_tarefa(id_tarefa);
* **parâmetros de entrada:**
  - id_tarefa: identificação interna de uma tarefa (valor devolvido pela função "cria tarefa"). Se NULL, especifica a tarefa corrente.
* **parâmetros de saída:**
  - TAREFA_NÃO_EXISTE,
  - DEADLOCK
  - OPERAÇÃO_OK
* **descrição:**
  Esta função desaloca uma estrutura de descrição de tarefa. Não é executada se não existirem outras tarefas a serem exe-

cutadas no núcleo. Caso a tarefa corrente tenha sido destruída, a mais prioritária em espera tomará o processador. A memória alocada durante a ação da tarefa sendo destruída não é liberada por esta função. É encargo da aplicação liberar os recursos alocados durante a operação da tarefa.

* **erros possíveis:**
  Erro se a tarefa referenciada não existe. Erro se é a última tarefa existente no núcleo.

### B.2.3 Reescalona

* **função:**
  Reordena as tarefas no núcleo segundo prioridades.
* **formato de chamada:**
  s = reescalona();
* **parâmetros de entrada:**
* **parâmetros de saída:** OPERAÇÃO_OK
* **descrição:**
  A função de reescalonamento faz com que seja possível que uma tarefa criada com maior prioridade tome forçadamente o processador, caso a tarefa ativa tenha menor prioridade. Não atua sobre tarefas suspensas.
* **erros possíveis:**
  *OBS:* Caso não haja outra tarefa presente no núcleo a operação desta função é nula. A tarefa ativa é colocada no estado "pronto", e o controle passa à tarefa mais prioritária em estado "pronto" antes desta inserção

### B.2.4 Suspende_tarefa

* **função:**

  A tarefa em execução corrente é suspensa, colocando-se a mesma na fila de "espera".

* **formato de chamada:**

  s = suspende_tarefa(timeout);

* **parâmetros de entrada:**

  · timeout: tempo associado à suspensão (em segundos)

* **parâmetros de saída:**

  · NO_MEM

  · DEADLOCK

  · OPERAÇÃO_OK

* **descrição:**

  A tarefa em execução é colocada em estado de suspensão. Seu descritor é atualizado convenientemente para indicação de seu novo estado. A função de suspensão permite que a tarefa corrente ceda sua vez no uso do processador. Uma tarefa fica em suspensão no aguardo de vencimento da temporização definida na entrada. A fila de tarefas suspensas não tem ordenação, uma vez que operações sobre a fila são feitas aleatoriamente.

* **erros possíveis:**

  Erro se a tarefa for a única presente no núcleo.

### B.2.5 Resume_tarefa

* **função:**

  Ativa uma tarefa para execuçao.

* **formato de chamada:**
  s = resume_tarefa(id_tarefa);
* **parâmetros de entrada:**
  id_tarefa : identificador da tarefa
* **parâmetros de saída:**
  - TAREFA_NÃO_EXISTE
  - OPERAÇÃO_OK
* **descrição:** Coloca um tarefa que estava na fila de "suspensos" na fila de tarefas a serem ativadas, permanecendo nesta fila com sua prioridade no momento da suspensão. Caso tenha prioridade maior que o processo corrente, provoca um reescalonamento e toma o processador.
* **erros possíveis:**
  Erro se não houver processo com a identificação da entrada.

### B.2.6 Muda_pri

* **função:**
  Modifica prioridade do processo.
* **formato de chamada:**
  s = muda_pri(id_tarefa,n_priv);
* **parâmetros de entrada:**
  - id_tarefa: identificação interna do processo
  - n_priv: nova prioridade do processo
* **parâmetros de saída:**
  - TAREFA_NÃO_EXISTE
  - PAR_INCORRETO
* **descrição:**

A função permite mudar a prioridade da tarefa especificada; a tarefa é colocada no estado "pronto" com a nova prioridade definida na entrada. A execução passa imediatamente à tarefa agora mais prioritária no núcleo, não havendo portanto retorno de resposta quando de operação correta.

* **erros possíveis:**
  Erro se a nova prioridade tem valor fora do possível no núcleo, ou se o processo especificado não existe.

### B.2.7 tarefa_ativa

* **função:**
  Identifica a tarefa ativa no momento.
* **formato de chamada:**
  tarefa_ativa(&id_proc);
* **parâmetros de entrada:**
  id_proc: área para cópia do descritor da tarefa ativa.
* **parâmetros de saída:**

* **descrição:**
  Identifica a tarefa que possui o processador, copiando para área do solicitante os dados que compõem o descritor da tarefa (ver anexo A - pag. 130). Usada em casos para obtenção de informação sobre a tarefa ativa ou para identificar qual esta tarefa em caso de pane no sistema.
* **erros possíveis:**
  -

## B.3 Gerência de sincronização e comunicação

### B.3.1 Cria porta

* **função:**

  Cria uma porta para comunicação entre tarefas.

* **formato de chamada:**

  s = cria_porta(qtdade_mens,tipo_porta, &id_porta);

* **parâmetros de entrada:**

  - qtdade_mens: quantidade de mensagens que poderá ser armazenada na porta.
  - tipo_porta: tipo da porta criada ("PORTA_NORMAL" ou "PORTA_ESPECIAL" ou "P_SEMÁFORO") [2]

* **parâmetros de saída:**

  - NO_MEM
  - PAR_INCORRETO
  - OPERAÇÃO_OK e "id porta" tem apontador para a porta criada.

* **descrição:**

  Cria uma porta, que é uma fila que vai conter mensagens enviadas para ela. A função cria esta fila, com o cabeçalho conforme o anexo A, ítem 4. A identificação da porta é um apontador para a própria fila criada.

[2] a porta de propósito especial é criada para efeito de suspensão de processos por tempo determinado ou com comportamento especial, agindo por exemplo como semáforos. No primeiro caso, a porta nunca recebe mensagens; no segundo caso, uma mensagem é uma permissão de acesso a um recurso controlado pelo semáforo. O uso deste tipo de porta pela aplicação é raro, e feito apenas em caso de necessidade de implementação de algum serviço especial, devido à particularidade de seu funcionamento.

* **erros possíveis:**
  Erro se não há mais memória para alocação da porta, ou se a quantidade de mensagens é maior que o especificado por configuração ou se o tipo da porta especificada não for um dos definidos (PORTA_NORMAL, PORTA_ESPECIAL).

### B.3.2 Espera_mens

* **função:**
  Busca na porta especificada uma mensagem.
* **formato de chamada:**
  s = espera_mens(id_porta,tempo_espera,tipo_verificação,&mens_obtida);
* **parâmetros de entrada:**
  - id_porta: apontador para a porta onde se busca a mensagem.
  - tempo_espera: tempo em segundos que a tarefa pode esperar por mensagem; se é zero, a tarefa não será suspensa. Não se aplica quando apenas se verifica existência de mensagem .
  - tipo_verificação: especifica se se deseja consumir a mensagem ou apenas verificar sua existência (selecionado entre valores "VERIFICA" e "CONSOME").
* **parâmetros de saída:**
  - TIMEOUT
  - DEADLOCK
  - PAR INCORRETO
  - OPERAÇÃO_OK e o número de mensagens em "mens_obtida" na porta se o parâmetro "tipo_verificação" for "verifica" ou um apontador para a a própria mensagem obtida, se

o parâmetro for "consome".

* **descrição:**

  Esta função permite que a tarefa ativa busque na porta especificada como parâmetro de entrada uma mensagem dirigida à ela, por um tempo especificado e com possibilidade de somente verificação sobre a existência de mensagem, sem consumo da mesma. Caso não exista mensagem na porta, a tarefa é suspensa, e o escalonador trata da eleição de outra para tomar o processador. Caso não exista outra tarefa ativa no ambiente, é gerado o erro de "DEADLOCK", pois a suspensão geraria deadlock. As tarefas são enfileiradas na porta, e quando chega uma mensagem a primeira tarefa a obtém. Esta fila não é ordenada por prioridade por facilidade e porque, em muitas aplicações, tarefas com prioridades diferentes não vão ficar na mesma porta.

* **erros possíveis:**

  Erro se a tarefa corrente é a única existente no núcleo e se suspensa levaria a "deadlock". Erro se ocorrer timeout na espera ou se o valor de entrada do timeout é 0 e não há mensagens à espera (não se aplica no caso de apenas verificação de existência de mensagens). Erro se o parâmetro "tipo_verifica" não for VERIFICA ou CONSOME.

### B.3.3 Obtém_mensagem

* **função:**

  Apanha a mensagem recebida em uma porta

* **formato de chamada:**

  s = obtem_mensagem(id_porta,&mens_obtida);

* **parâmetros de entrada:**

- id porta: apontador para a porta onde se vai ler a mensagem.

* **parâmetros de saída:**
  - NO_MENS
  - PAR_INCORRETO
  - OPERAÇÃO_OK e a mensagem obtida em "mens_obtida"
* **descrição:**

  Esta função obtém uma mensagem da porta especificada, caso haja. A chamada anterior de "espera_mens" para verificação de existência de mensagem pode ser feita, e esta função usada para apenas obter a mensagem, caso desejada.
* **erros possíveis:**

  Erro se não existe a porta especificada na entrada ou se não existe mensagem nesta porta.

### B.3.4 Envia_mens

* **função:**

  Envia uma mensagem para uma determinada porta.
* **formato de chamada:**

  s = envia_mens(id_porta, apt_mens);
* **parâmetros de entrada:**
  - id_porta: apontador para a porta que vai receber a mensagem.
  - apt_mens: apontador da mensagem que vai ser inserida na porta.
* **parâmetros de saída:**
  - PAR_INCORRETO
  - PORTA_OCUPADA

- OPERAÇÃO_OK

* **descrição:**

  Esta função provoca a colocação de uma mensagem, apontada pelo dado de entrada, em uma determinada porta. Se quando da colocação da mensagem na porta há uma tarefa à espera, a mensagem que está na primeira posição da fila de mensagens da porta é entregue à tarefa, e a tarefa é colocada no estado "pronto". Se esta tarefa for ativada, a tarefa corrente vai ser suspensa. Isto ocorre apenas se a prioridade da tarefa que recebeu a mensagem for menor que aquela que enviou a mensagem. Para que uma tarefa de menor prioridade receba uma mensagem, a tarefa que envia (maior prioridade) deve ceder o processador via sua suspensão ou chamando a função de reescalonamento.

* **erros possíveis:**

  Erro se não existe a porta especificada na entrada ou se a porta especificada é do tipo especial ou se a fila associada à porta está cheia.

### B.3.5 Libera_porta

* **função:**

  Termina a existência de uma porta no núcleo.

* **formato de chamada:**

  s = libera porta(id porta);

* **parâmetros de entrada:**

  - id_porta: apontador para a porta a ser deletada.

* **parâmetros de saída:**

  - PAR_INCORRETO

- PORTA OCUPADA
- OPERAÇÃO OK

* **descrição:**

  Esta função permite terminar a existência de uma determinada porta no núcleo, deste que a mesma esteja vazia (não haja mensagens nem tarefas à espera na porta).

* **erros possíveis:**

  Porta não existente ou porta com tarefas suspensas à espera de mensagem.

## B.4 Gerência de temporizações

[3]

### B.4.1 Cria_temp

* **função:**

  Aloca uma célula de temporização ao processo que solicita a funcão, de acordo com os parâmetros de entrada

* **formato de chamada:**

  s = Cria_temp(max_temp,tipo_temp,tipo_cel);

* **parâmetros de entrada:**

  - max temp: máxima temporização, em segundos, que a célula deve "contar"
  - tipo_temp: tipo de temporização (única ou cíclica).
  - tipo_cel: tipo da célula: ligada ao núcleo (relógio de tempo real), ligada a suspensão de processo, ligada a semáforo, ligada a porta.

[3] não implementada; ver considerações no capítulo 4

* **parâmetros de saída:**
  - TE_NOMEM
  - saída comtém "id temp", identificação da célula de temporização criada.
  - MAX_TEMP: máximo número de temporizações foi atingido
* **descrição:**

  Esta função permite a criação de uma célula de temporização . As inicializações são efetuadas, como a identificação da célula, validação do descritor de temporização, o valor máximo de segundos que deve contar e o tipo de temporização associada: única ou cíclica, conforme a estrutura de descritor de temporização (ver anexo 3). A temporização única implica em que, quando há o vencimento da temporização, os processos associados à ela são ativados e a célula de temporização retirada do sistema. A temporização cíclica coloca os processos como ativos e a célula é re-inserida no sistema, com o valor de contagem reiniciada. Há na estrutura da célula de temporização um contador de quantas vezes isto foi feito. Após a criação de temporização começa-se imediatamente a contar o tempo associado. Esta função se insere no contexto de entidade básica, não podendo ser diretamente chamada pela aplicação. A utilização desta função deve ser vista no contexto de existência de células de temporização ligadas a vários tipos de serviços.
* **erros possíveis:**

  Erro se não há mais memória para a alocação de células de temporização ou se o máximo número de temporizações foi atingido.

### B.4.2 Desconecta_temp

* **função:**
  Invalida uma célula de temporização.
* **formato de chamada:**
  s = desconecta temp(id temp,tipo desconexão);
* **parâmetros de entrada:**
  - id_temp: identifica a célula de temporização a ser cancelada.
  - tipo_desconexão: se verdadeiro, cancela a célula de temporização, colocando quaisquer processos associados a ela no estado "pronto". Se falso, não cancela a célula de temporização caso haja processos a ela associados.
* **parâmetros de saída:**
  - NO_TEMP_DEF
  - TEMP_OCUPADA
  - FALTA_PRIV
  - OPERAÇÃO_OK
* **descrição:**
  Esta função permite que se invalide uma célula de temporização existente. No caso de uma temporização associada a processo, se há processos associados (de acordo com o parâmetro "tipo_desconexão"), os mesmos são colocados na condição de execução ("prontos"). Uma temporizaçao do tipo de contagem de tempo real não pode ser cancelada.
* **erros possíveis:**
  Erro se a célula não existe ou existem processos suspensos pela temporização (de acordo com o tipo de temporização e com parâmetro "tipo desconexão") ou se for uma tempo-

rização de marcação de tempo real.

### B.4.3 Controla_temp

* **função:**
  Controla a operação da temporização associada à célula especificada.
* **formato de chamada:**
  s = controla_temp(id_temp, condição,tipo)
* **parâmetros de entrada:**
  - id_temp: identifica a célula de temporização sobre a qual se opera.
  - condição: indica a operação a ser efetuada sobre a temporização, que pode ser congelamento ou reiniciação
  - tipo: se verdadeiro, especifica que, se houver processos associados, não pode ser executada qualquer operação sobre a célula de temporização.
* **parâmetros de saída:**
  - NO_TEMP_DEF
  - TEMP_OCUPADA
  - TE_NOERR
* **descrição:**
  Esta função permite a operação sobre uma determinada temporização através da alteração do status da célula de temporização associada. No caso de "tipo" especificar que isso pode ser feito mesmo que haja processos suspensos em temporização. O congelamento implica em que a célula terá tratamento normal no tocante à atualização de tempos, mas, quando do vencimento do tempo associado será re-inserida

na fila, como no caso de uma temporização cíclica. O congelamento não é possível em temporizações cíclicas (como por exemplo a marcação de tempo real).

* **erros possíveis:**
  Erro se não existe a célula especificada ou se existem processos associados e o parâmetro "tipo" não autoriza modificações ou se a temporização representada pela célula é cíclica.

### B.4.4 Le_temp

* **função:**
  Leitura do contador associado temporização.
* **formato de chamada:**
  s = le temp(id temp,&contador,&ciclos);
* **parâmetros de entrada:**
  - id temp: identifica a célula de temporização associada ao contador que se deseja verificar.
* **parâmetros de saída:**
  - OPERAÇÃO OK, e contador: valor atual do contador associado à célula de temporização (em segundos), ciclos: quantidade de vezes que a temporização já venceu (caso que ocorre quando a temporização é cíclica.
  - NO TEMP DEF
* **descrição:**
  Esta função permite que se obtenha o valor do contador interno de temporização de uma célula e o número de vezes que foi reiniciada a temporização, no caso cíclico.
* **erros possíveis:**

Erro se não existe a célula especificada.

## B.5 Gerência de memória

### B.5.1 Pede_mem

* **função:**
  Solicita memória .
* **formato de chamada:**
  s = pede_mem(qtdade,&pt_mem)
* **parâmetros de entrada:**
  - qtdade: quantidade em bytes de memória a ser alocada
* **parâmetros de saída:**
  - NO_MEM
  - OPERAÇÃO_OK, e em "pt_mem" o apontador para área alocada de tamanho maior ou igual a "qtdade" bytes
* **descrição:**
  Esta função aloca uma quantidade de memória para uma tarefa; o tamanho em bytes alocado pode ser aproximado para cima.
* **erros possíveis:**
  Erro se não há memória disponível

### B.5.2 Libera_mem

* **função:**
  Libera memória previamente alocada pela função "pede_mem".
* **formato de chamada:**

s = libera_mem(pt_mem);

* **parâmetros de entrada:**
  - pt_mem: apontador para a área alocada anteriormente.
* **parâmetros de saída:**
  - PAR_INCORRETO
  - OPERAÇÃO OK
* **descrição:**
  Libera blocos de memória alocados pela função "pede_mem".
* **erros possíveis:**
  Erro se a área referenciada não foi anteriormente alocada (ponteiro incorreto).

### B.5.3 Mem_disp

* **função:**
  Informa quanta memória está disponível no núcleo.
* **formato de chamada:**
  s = mem_disp( &total_mem )
* **parâmetros de entrada:** -
* **parâmetros de saída:**
  - OPERAÇÃO_OK e "total_mem" contém a quantidade aproximada em bytes de memória disponível.
* **descrição:**
  A função devolve o número de bytes disponíveis para alocação.
* **erros possíveis:** -

## B.6 Gerência de semáforos

### B.6.1 Cria semáforo

* **função:**
  Aloca área para semáforo
* **formato de chamada:**
  s = cria_semáforo(tamanho,&id_sem)
* **parâmetros de entrada:**
  - tamanho: número de acessos permitidos ao recurso que o semáforo controla. Para um semáforo binário, tem-se "tamanho" com o valor 1.
* **parâmetros de saída:**
  - NO_MEM
  - PAR_INCORRETO
  - OPERAÇÃO_OK e "id_sem" contém um apontador para o semáforo alocado
* **descrição:**
  A criação de um semáforo, nesta implementação, equivale a criar uma "porta" de um tipo especial, contendo uma fila de mensagens do tamanho equivalente à contagem do semáforo. Cada mensagem equivale a uma "permissão" de acesso a um recurso associado ao semáforo. Um semáforo binário tem uma fila associada de tamanho 1, ou seja, existe apenas permissão para acesso de uma tarefa por vez a um recurso a ele associado.
* **erros possíveis:**
  Erro se não houver área para alocação do semáforo, ou se houver parâmetro incorreto ("tamanho" menor ou igual a

0).

*obs*: o parâmetro de definição de quantidade máxima de portas existente no núcleo deve incluir também a quantidade de semáforos a serem criados.

### B.6.2 Suspende_semáforo

* **função:**
  Associa uma tarefa a um semáforo
* **formato de chamada:**
  s = suspende_semáforo(id_sem)
* **parâmetros de entrada:**
  - id_sem: apontador para o semáforo a ser utilizado
* **parâmetros de saída:**
  - OPERAÇÃO_OK
  - DEADLOCK
  - PAR INCORRETO
  - TIMEOUT
* **descrição:** Se o semáforo existe e se houver condição para acesso ao semáforo, o processo solicitante da função pode continuar sua execução, que será provavelmente o acesso a um recurso compartilhado. Caso contrário, o processo é suspenso e o próximo processo de maior prioridade que estava suspenso vai assumir o processador. O processo solicitante permanecerá suspenso pelo primeiro evento a acontecer: o tempo máximo especificado por configuração ou a chegada de uma "permissão" para continuar; após uma destas ocorrências, será novamente ativado, com informação do "timeout" ocorrido. Caso não haja processo pendente e o

atual seja o único presente no núcleo, este não será suspenso devido à ocorrência certa de deadlock.

* **erros possíveis:**
  Erro se o semáforo referido não existe ou se o processo ativo é o único presente no núcleo, ou ainda se houve estouro de tempo na espera pelo semáforo.

### B.6.3 Sinaliza_semáforo

* **função:**
  Devolve ao semáforo uma informação que dá direito de acesso ao mesmo. Libera eventual processo à espera.
* **formato de chamada:**
  s = sinaliza_semáforo(id_sem)
* **parâmetros de entrada:**
  - id_sem: apontador para o semáforo a ser sinalizado
* **parâmetros de saída:**
  - OPERAÇÃO_OK
  - PAR_INCORRETO
* **descrição:**
  A função devolve ao semáforo uma permissão para futuros acessos ao recurso por ele controlado. Se há algum processo suspenso à espera desta sinalização, o mesmo vai ser colocado na fila de "prontos" . A tarefa que executa esta função deve se suspender para permitir que outra tarefa, de menor prioridade, possa obter o processador.
* **erros possíveis:**
  Erro se não existe o semáforo identificado ou se é sinalizado excessivamente um semáforo (número de sinalizações maior

que o tamanho com que o mesmo foi criado)

### B.6.4 Deleta semáforo

* **função:**
  Desaloca uma área de semáforo
* **formato de chamada:**
  s = deleta_semáforo(id_sem)
* **parâmetros de entrada:**
  - id_sem: apontador para o semáforo a ser deletado
* **parâmetros de saída:**
  - OPERAÇÃO OK
  - SEM_OCUPADO,
  - PAR INCORRETO.
* **descrição:**
  A função verifica se não há processos suspensos no semáforo e, se não houver, libera a área do semáforo.
* **erros possíveis:**
  Erro se o semáforo não existe ou se há processos suspensos à espera de sinalização no semáforo.

# Bibliografia


[1] W. Heath. A system executive for real-time microcomputer programs. *IEEE Micro*, junho 1984.

[2] J. White J. Wright. Real-time operating systems and multitask programming.

[3] T. Parker. An introduction to real-time systems. *Computer Language*, july 1989.

[4] J. Isaak. Designing systems for real time applications. *Byte*, april 1984.

[5] G. Elfring. A guide to real-time executives. *Computer Language*, 1986.

[6] J. McQuaid. Real-time scheduler keeps jobs moving. *Research and Development*, dezember 1985.

[7] T. Ess. Rrte: a commercial embedded product real time executive.

[8] R. Paul I. Lee, R. King. A predictable real-time kernel for distributed multisensor systems. *Computer*, june 1989.

[9] D. Simpson. Real-time riscs. *Systems Integration*, july 1989.

[10] D. Cheriton et Al. Thoth, a portable real-time operating system. *Communications of the ACM*, 22(2), FEBRUARY 1979.

[11] M. Agoston. Microprocessor operating system: the kernel.

[12] R. Holt et Al. *Structured concurrent programming with operating systems applications.* Addison-Wesley, 1978.

[13] R. Pressman. *Software Engineering - a practitioner's approach.* McGraw-Hill, 1982.

[14] F. Halsall et Al. Development environment for the design and test of applications software for a distributed multiprocessor computer system. *IEE Proc. - part E*, 130(1), january 1983.

[15] M. Fox. An organizational view of distributed systems. *IEEE trans. on systems, man and cybernetics*, smc-11(1), january 1981.

[16] H. Napiatek J. Gorski. Formal specification of basic mechanisms of a message passing kernel. *Microprocessing and Microprogramming*, 21, 1987.

[17] D. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), december 1972.

[18] J. B'ezivin et Al. A methodological approach to the design of real-time operating systems.

[19] A. Tannenbaum. *Operating Systems: Design and implementation.* Prentice-Hall, 1987.

[20] A. Silberschatz J. Peterson. *Operating Systems Concepts - 2nd Edition.* Addison-Wesley, 1985.

[21] Donovan Madnick. *Operating Systems.* McGraw-Hill, 1974.

[22] D. Comer. *Operating System Design - the XINU approach.* Prentice-Hall, 1984.

[23] A. Jones. The object model: a conceptual tool for structuring software.

[24] B. Cox. *Object-Oriented Programming - an evolutionary approach.* Addison-Wesley, 1986.

[25] J. Kramer et Al. Conic: an integrated approach to distributed computer control systems. *IEE Proc. - part E*, 130(1), january 1983.

[26] W. Bo K. Schwan, P. Gopinath. Chaos: kernel support for objects in the real-time domain. *IEEE trans. on computers*, C-36(8), AUGUST 1987.

[27] A. Peltomaa H. Koivo. Microcomputer real-time multitasking operating systems in control applications. *North-Holland Computers in Industry*, 5, 1984.

[28] Ritchie Kernighan. *The C programming language.* Addison-Wesley, 1984.

[29] T. Norman J. Peterson. Buddy systems. *Comm. of the ACM*, 20(5), june 1977.

[30] V. Milutinovic B. Furht. A survey of microprocessor architectures for memory management. *Computer*, march 1987.

[31] H. Azevedo. Escalonadores de tempo real. 1989.

[32] A. Holub. Priority queues. *Dr. Dobbs Journal*, june 1987.

[33] P. Conte. Recycle your software. *Computer Language*, june 1988.

[34] I. Ribeiro H. Almeida. *NTR 80 - nucleo de tempo real, versao 5.0.* Technical Report, CTI/IA/DCP/ETR, 1986.

[35] L. Nieuwenhuis. Mirtex: a real-time, multi-tasking executive for microprocessors. *Microprocessing and Microprogramming*, 12, 1983.

[36] M. Krieger E. Fathi. An executive for task-driven multimicrocomputer systems. *IEEE Micro*, october 1983.

[37] I. Kovashki L. Nikolov. Design and implementation of a portable kernel for real-time applications. *Microprocessing and Microprogramming*, 21, 1987.

[38] G. Schoder T. Bemmmerl. A portable realtime multitasking kernel for embedded microprocessor systems. *Microprocessing and Microprogramming*, 21, 1987.

[39] T. Wicklund. Mini-exec: a portable executive for 8-bit microcomputers. *Communications of the ACM*, 28(11), november 1982.

[40] M. Schindler. Toward faster and portable real-time operating systems. *Eletronic Design*, january 1988.

[41] R. Scott. Rolling your own. *Embedded systems programming*, february 1989.

[42] G. Schrott. Generation of dedicated realtime operating systems by dialogue.

[43] A. Kvitca. *Resolution de problemas con inteligencia artificial.* Ed. EBAI, 1988.

[44] J. Dlugosz. A multitasking kernel for c programmers. *Computer Language*, october 1988.

[45] D. Bowling. Real-time modeling with ms-dos. *Dr. Dobbs Journal*, february 1989.

[46] D. Jacintho. Projeto SISDI/MAP - tese de mestrado em fase de terminacao. 1989.

[47] M. Zabeu et Al. Protocolo rs511 - um modelo de implementacao. In *7. SBRC*, 1989.

[48] M. Zabeu et Al. Sisdi map - sistema did'atico do protocolo e da interface de aplicacao mms do map. In *Seminario franco-brasileiro em sistemas informaticos distribuidos*, 1989.

[49] A. Holub. A preemptive multitasking kernel and more mean subroutines. *Dr. Dobbs Journal*, december 1987.

[50] A. Holub. A preemptive multitasking kernel continued, lattice dbc and compiler controversies. *Dr. Dobbs Journal*, january 1988.

[51] A. Holub. Stalking the wild memory allocator. *Dr. Dobbs Journal*, june 1988.

[52] I. Ashdown. Dynamic multidimansional arrays in c. *Computer Language*, june 1988.

[53] Standard specification for microprocessor operating systems interfaces - ieee task 855. 1983.

[54] JH. Conwan D. Jackson. The proposed ieee 855 microprocessor operating systems interface standard. *IEEE Micro*, august 1984.

[55] S. Tonami M. Aoki, T. Uchiyama. Protocol processing for high-speed packet switching systems. 1987.

[56] T. Sakuma T. Ozeki, M. Matsushita. Implementation of real-time operating system for integrated switching system. 1988.

[57] L. Read A. Adams. Fast packet technology for intelligent netword applications.

[58] J. Bunch A. Adams. A software architecture for a next generation switching system.

[59] 'Area de Sistemas/DCT. *Uma proposta para modelagem de implementa,c ao de sistemas.* Technical Report, CPqD - Telebr'as, 1989.

[60] F. Halsal. *DATA Communications, Computer Networks and OSI.* Addison-Wesley, 1988.

[61] A. Bache et Al. Implementation and applications of the scipion packet switching node. 88.

[62] T. Uchiyama H. Ichikawa, M. Aoki. High-speed packet switching systems for multimedia communications. *IEEE journal on selected areas in communications*, SAC-5(8), october 1987.

[63] A. Miller R. Dirvin. The mc 68824 token bus controller. *IEEE Micro*, june 1986.

[64] TBC - MC 68834 (Token-passing Bus Controller). Motorola technical data - preliminary technical summary.

[65] T. Ohrui. The dex-ctron operating system applicable to isdn switching and communications processing systems. 1988.