1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-4149-TDL/267		*Abril 1987*	☐ Interna ☒ Externa ☐ Restrita

4. Origem	Programa
ECO/SDA	*PG*

6. Palavras chaves - selecionadas pelo(s) autor(es)

MEMÓRIA DE MASSA
SISTEMA DE ARQUIVOS
E/S DE ALTO DESEMPENHO
ESCALONAMENTO DE E/S
MEMÓRIA "CACHE"

7. C.D.U.: *629.783:621.39*

8. Título *INPE-4149-TDL/267*

MONITOR DE DISCO DE ALTO DESEMPENHO PARA APLICAÇÕES EM TEMPO REAL

10. Páginas: *214*

11. Última página: *B.10*

12. Revisada por

Ricardo Z.
Ricardo C. O. Martins

9. Autoria *Maurício Macedo de Faria*

Assinatura responsável

13. Autorizada por

Marco Antonio Raupp
Diretor Geral

14. Resumo/Notas

Este trabalho apresenta os conceitos teóricos envolvidos no projeto e na implementação de um programa Monitor de Disco de Alto Desempenho. Apresenta também uma proposta de implementação, que detalha os aspectos relevantes necessários à obtenção de alto desempenho, simplicidade e portabilidade.

15. Observações

Dissertação de mestrado em Eletrônica e Telecomunicações aprovada em junho de 1986.

Aprovada pela Banca Examinadora
em cumprimento a requisito exigido
para a obtenção do Título de Mestre
em Eletrônica e Telecomunicações

Dr. Eduardo Whitaker Bergamini — Presidente

Dr. Ricardo Corrêa de O.Martins — Orientador

Dr. Marcio Luiz de Andrade Netto — Membro da Banca -convidado-

Dr. Alderico Rodrigues de Paula Jr. — Membro da Banca

Engº Wilson Yamaguti, Mestre — Membro da Banca

Candidato: Maurício Macedo de Faria

São José dos Campos, 18 de junho de 1986

DEDICATÓRIA

A minha esposa, cuja ajuda e compreensão foram essenciais para a realização deste trabalho.

AGRADECIMENTOS

Agradeço ao Dr. Ricardo C.O. Martins pela sua orientação; aos membros da banca, em especial ao Dr. Mário L.A. Netto do Centro Tecnológico para Informática - CTI; e a todos os amigos que contribuíram para a realização deste trabalho.

ABSTRACT

This work presents the theoretical concepts involved in the design and implementation of a High Performance Disk Monitor Program. An implementation proposal detailing the relevant aspects required to obtain high performance, simplicity and portability is also presented.

SUMÁRIO

Pág.

Pág.

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

MONITOR DE DISCO DE ALTO DESEMPENHO PARA APLICAÇÕES EM TEMPO REAL

1.1 - INTRODUÇÃO

Este trabalho apresenta um Monitor de Disco de alto desempenho para aplicação em sistemas de controle de processos em tempo real multiprogramados, onde é necessário o uso de discos magnéticos para o armazenamento de uma grande quantidade de informações.

O Monitor de Disco cria uma estrutura lógica composta de diretórios, arquivos e registros, que é mapeada para a estrutura física da máquina. A estrutura física compõe-se dos discos magnéticos com suas trilhas e setores físicos.

Os vários processos de aplicação podem então utilizar os discos magnéticos de uma forma lógica, independentemente de suas características físicas.

Os principais objetivos do Monitor de Disco são, em ordem de importância:

1) *Desempenho:* os recursos físicos devem ser utilizados de maneira eficiente, de modo a adequá-los as necessidades dos processos em tempo real.

2) *Simplicidade:* a estrutura lógica criada pelo monitor deve ser versátil. Porém, ela deve se manter simples para não comprometer o desempenho.

3) *Configuração:* o monitor deve ter seus parâmetros de operação configuráveis para adequar-se às várias aplicações e implementações físicas distintas.

4) *Portabilidade:* o monitor deve possuir uma estrutura modular, de fácil modificação e adaptação a vários ambientes, sem prejuízo do desempenho.

O Monitor de Disco possui as seguintes interfaces:

a) com os usuários que vão configurar o monitor, realizada através de primitivas de configuração.

b) com os processos de aplicação, realizada através de primitivas de manipulação implementadas pelo monitor.

c) com os dispositivos físicos, realizada através de rotinas de acesso que utilizam primitivas e procedimentos existentes no núcleo do sistema operacional da máquina.

O Monitor de Disco adota uma estrutura hierárquica similar àquela proposta por Dijkstra (1968) e citada em Madnick e Aslop (1969). O aspecto mais importante dessa organização é que todas as atividades podem ser divididas em atividades realizadas sequencialmente.

A estrutura hierárquica dessas atividades sequenciais resulta numa organização em níveis, cujo conceito é conhecido como "modularização hierárquica".

A estrutura hierárquica utilizada é composta de seis níveis descritos a seguir, conforme mostra a Figura 1.1.

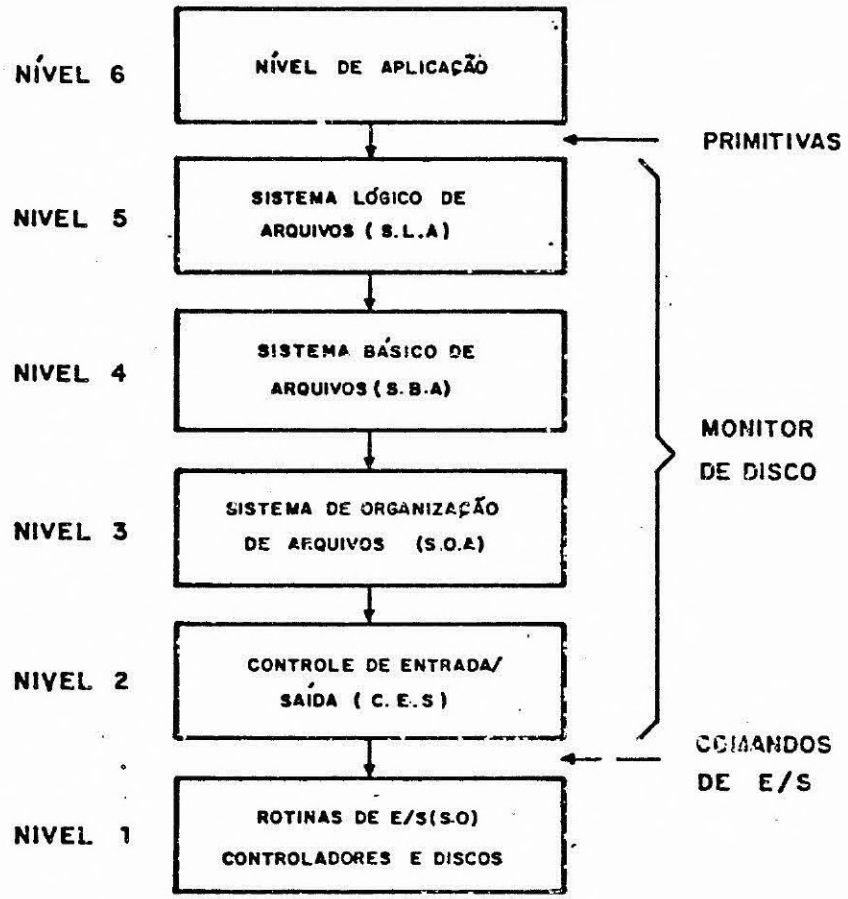


Fig. 1.1 - Estrutura hierárquica.

O nível 6 é formado pelos usuários que vão configurar o monitor para as várias aplicações e pelos processos de aplicação que vão utilizar os arquivos.

O nível 5 permite que os processos acessem os arquivos através de nomes simbólicos, os quais são mantidos em diretórios associados aos dispositivos periféricos.

O nível 4 controla os arquivos, seu tamanho, conteúdo e também faz a verificação dos tipos de acesso para a proteção das informações.

O nível 3 faz o mapeamento da estrutura lógica dos arquivos para a estrutura física, já considerando as características físicas dos meios de armazenamento. É responsável também pelo gerenciamento do espaço do disco.

O nível 2 faz a interface do monitor com o "hardware". Tem por função fazer o uso eficiente dos dispositivos físicos, implementando rotinas de otimização adequadas.

O nível 1 é formado pelas rotinas de Entrada/Saída - E/S ("drivers"), pelos controladores e pelos discos magnéticos.

O escopo do trabalho abrange os níveis 2, 3, 4 e 5. Apresenta-se em seguida uma breve descrição do conteúdo de cada um dos próximos capítulos.

No Capítulo 2 são apresentados os conceitos básicos envolvidos no trabalho. É feito um detalhamento de todos os tópicos relacionados ao projeto do Monitor de Disco, expondo as alternativas existentes e situando-as de acordo com o objetivo proposto. São abordados ainda os conceitos de tipos de arquivos e sua organização física, os sistemas de diretório, as operações básicas em arquivos, a alocação do espaço da memória secundária, a proteção de arquivo, assim como as estratégias e algoritmos utilizados para aumentar o desempenho.

No Capítulo 3 é formulada a proposta do Monitor de Disco baseada na estrutura hierárquica já citada. É feito um detalhamento dos níveis da estrutura hierárquica, salientando suas funções, as estruturas de dados de cada nível e as interações entre os vários níveis. Em seguida é apresentada a seleção das técnicas a serem adotadas no trabalho em função das alternativas descritas no Capítulo 2, visando obter o melhor desempenho possível. É proposto também o conjunto de primitivas para o monitor.

No Capítulo 4 é feito o detalhamento da proposta apresentada no Capítulo 3. São apresentados diversos detalhes de implementação das técnicas adotadas, além dos algoritmos utilizados. Em seguida, é feito o detalhamento das primitivas com seus parâmetros e estruturas de dados afetadas. É mostrado o modo de implementação do monitor, especificando a interface com o núcleo do sistema operacional da máquina.

O Capítulo 5 apresenta as conclusões do trabalho, indicando o caminho para futuros aperfeiçoamentos.

Finalizando, os apêndices apresentam as primitivas e as rotinas de otimização na forma de pseudocódigo de alto nível.

CAPÍTULO 2

CONCEITOS BÁSICOS

2.1 - INTRODUÇÃO

Neste capítulo são apresentados os aspectos teóricos relacionados com monitores de discos.

Em primeiro lugar é apresentado o conceito de arquivos, os tipos básicos existentes e a necessidade de diferentes organizações físicas. Em seguida, são analisados os métodos de organização de arquivos baseados em algumas características principais das três organizações básicas: sequencial, aleatória e do tipo lista.

Depois são abrangidos os sistemas de diretório, seus tipos e suas estruturas de implementação, além das operações básicas que podem ser feitas para o acesso aos arquivos.

Apresentam-se a seguir os métodos de proteção de arquivos contra os acessos indevidos, bem como os de alocação do espaço do disco e os de gerenciamento do espaço livre.

Finalmente, para que se obtenha um alto desempenho é necessário que sejam implementados técnicas de otimização que aproveitem ao máximo as características físicas dos dispositivos de armazenamento utilizados pela máquina.

2.2 - CONCEITOS DE ARQUIVOS

Um arquivo é uma estrutura lógica de armazenamento definida pelo Sistema Operacional, de modo a permitir que o acesso às informações independa das características físicas dos dispositivos de armazenamento utilizados.

Normalmente um arquivo é uma coleção de informações relacionadas entre si sendo constituído de vários registros ("records"). Os registros são as unidades de informações, as quais são acessadas de cada vez.

Existem registros lógicos e registros físicos. Em geral, os registros físicos refletem as características dos dispositivos de armazenamento e são de um tamanho de tal forma que permitam o acesso mais eficiente possível. Um registro lógico pode ser formado por vários registros físicos ou vice-versa.

2.2.1 - TIPOS DE ARQUIVO

Os arquivos podem armazenar vários tipos de informações: dados numéricos, textos, folhas de pagamento, programas fonte, programas objeto etc. Cada arquivo possui uma determinada "estrutura" de acordo com o seu uso. Por exemplo, um arquivo de textos é uma sequência de caracteres organizados em linhas e páginas; um arquivo de programa fonte é uma sequência de linhas que representa sub-rotinas e funções, cada uma organizada em declarações e comandos executáveis; um arquivo de programa objeto é uma sequência de palavras organizadas em blocos para serem carregados na memória (Peterson and Silberschatz, 1983).

Como ressalta Peterson e Silberschatz (1983), um aspecto importante a ser visto aqui é: Qual o tipo de controle que o Sistema Operacional deve possuir sobre a estrutura e o tipo dos arquivos?

Caso o Sistema Operacional conheça a estrutura de um arquivo, ele pode operar sobre este arquivo executando vários serviços e evitando erros de manipulação. Isso alivia várias tarefas por parte do usuário mas, por outro lado, torna o Sistema Operacional mais complexo e mais extenso, além de não permitir o uso de novas estruturas de arquivos que não foram previstas anteriormente.

O outro extremo a ser considerado é quando o Sistema Operacional não impõe (nem suporta) nenhuma estrutura ou tipo de arquivo predefinidos, como é o caso do UNIX (Ritchie and Thompson, 1974). Neste caso, não é feita nenhuma interpretação da informação contida no arquivo, o que permite a máxima flexibilidade mas não fornece nenhum suporte ao usuário, o qual deve se preocupar com todos os detalhes.

2.2.2 - ORGANIZAÇÃO FÍSICA

As várias aplicações dos computadores fazem com que existam diferentes necessidades dos usuários ou dos processos em relação ao armazenamento, recuperação e atualização das informações contidas no disco.

Em algumas situações é necessário que os dados sejam recuperados rapidamente, mas a atualização dos dados já existentes pode ser feita mais lentamente (por exemplo: sistemas de bancos de dados). Em outras situações é importante que os dados sejam armazenados e recuperados rapidamente, não existindo a preocupação com a utilização eficiente do espaço do disco (por exemplo: sistemas de controle em tempo real).

Portanto, cada tipo de aplicação requer um tipo de organização dos dados que atenda suas necessidades e, além disto, faça uso eficiente do "hardware", causando a mínima sobrecarga possível ao sistema (Dodd, 1969).

Nesse caso, as diferentes organizações de dados são utilizadas para "moldar" os requisitos lógicos do usuário as características físicas dos meios de armazenamento e do "hardware" da máquina (Dodd, 1969).

2.2.3 - MÉTODOS DE ORGANIZAÇÃO DE ARQUIVOS

As necessidades do uso de uma determinada organização dos arquivos levam à escolha de uma organização que melhor se adapte à aplicação. Esta escolha pode ser baseada na observação de quatro características das diferentes organizações:

- naturalidade,
- acesso,
- manutenção,
- suporte.

A *naturalidade* é a relação entre a organização do arquivo e a dos dados. É um ponto parcialmente subjetivo e dependente da familiaridade e facilidade de entendimento por parte do ser humano.

As pessoas preferem as técnicas de organização que elas entendam e tenham tido alguma experiência anterior (Chapin, 1969).

O *acesso* a um arquivo é geralmente feito de três maneiras. A primeira delas é o acesso por "atributo", normalmente usado para encontrar um valor, o que pode também envolver a soma lógica de vários atributos. É o modo de acesso que tem dominado as técnicas de organização de arquivos.

A segunda maneira de acesso é por "propriedade", normalmente usada para encontrar outras propriedades ou valores. Este tipo de acesso utiliza tanto atributos como valores no processo de acesso e normalmente envolve a soma lógica de tais características.

A terceira maneira de acesso é por "valor", usada para encontrar o atributo correspondente. É a forma menos comum das maneiras de acesso e, na prática, é quase sempre seguida de acessos adicionais por atributo ou por propriedade.

A *manutenção* de arquivos consiste fundamentalmente em três operações: adicionar novas informações ao arquivo, alterar o conteúdo existente no arquivo, isto é, modificar os valores em algumas propriedades, mas não nos atributos, e remover informações do arquivo.

Embora a escolha de uma técnica de organização de arquivos dependa em grande parte do tipo de acesso e do tipo de manutenção que se deseja realizar, na prática esta escolha fica dependendo do *suporte* de "hardware" e "software" existentes, pois a escolha deve ser baseada no fato de que a técnica selecionada deve funcionar adequadamente com o "software" e "hardware" disponíveis na máquina (Chapin, 1969).

Verifica-se que todos os métodos de organização apresentados podem ser elaborados a partir de três organizações básicas (Dodd, 1969):

- sequenciais,
- aleatória,
- tipo lista.

São descritas e comparadas a seguir as organizações básicas em relação às quatro características citadas acima.

2.2.4 - ORGANIZAÇÕES SEQUENCIAIS

A organização sequencial é talvez o tipo mais conhecido de organização de arquivos. Os vários registros são armazenados em posições relativas a outros registros, de acordo com uma sequencia especificada. Para ordenar os registros em sequência, um atributo comum dos registros, chamado de "chave", é escolhido.

Os registros também podem ser armazenados sem uma chave. Este caso ocorre quando os registros são armazenados na ordem de sua chegada ao sistema. Em qualquer caso, a ordem lógica dos registros no arquivo e a ordem física dos registros no meio de armazenamento é a mesma (Dodd, 1969).

A grande *vantagem* da organização sequencial é o rápido acesso por atributo durante a recuperação. Se o atributo utilizado na criação do arquivo for o mesmo usado para a recuperação, o acesso ao próprio registro torna-se mais rápido. Porém, para a procura de um registro qualquer com uma determinada chave, a organização sequencial torna-se muito lenta.

A atualização de valores numa organização sequencial é difícil, pois, caso o novo registro a ser armazenado seja maior ou menor que o registro original, os registros adjacentes têm de ser reescritos para permitir a atualização. A atualização de um único registro torna-se muito rara e somente é feita quando existe um grande número de registros para serem atualizados. Tais organizações sequenciais tipicamente necessitam de reordenação frequente, o que causa um custo maior na manutenção.

Uma variação da organização sequencial é a *sequencial indexada*, na qual são adicionados índices às chaves para permitir o acesso por atributo, quando o atributo é parte da chave usada na ordenação do arquivo.

Nesse caso então, os registros podem ser acessados através do uso de índices ou sequencialmente. A Figura 2.1 mostra a organização sequencial indexada.

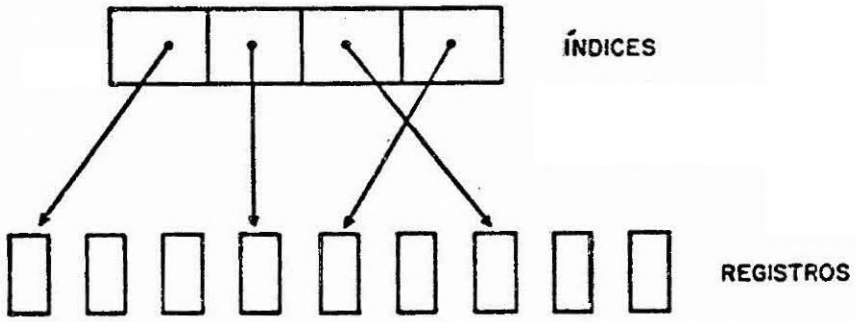


Fig. 2.1 - Organização sequencial indexada.

Geralmente os índices são orientados para as características dos meios de armazenamento, o que leva a uma redução do tempo gasto para "percorrer" o arquivo. Isto permite o acesso a qualquer componente do arquivo, dada a chave, gastando o mesmo tempo. (Chapin, 1969).

A forma mais simples para um arquivo sequencial indexado é aquela na qual o índice contém o endereço de todos os registros de dados. É chamada de indexação completa ("full indexing") e possibilita um acesso muito rápido a qualquer registro do arquivo. Entretanto, se o arquivo é muito grande, o índice também torna-se extenso e a perda de espaço fica muito grande.

Por essa razão, utiliza-se mais de um nível de índices de forma hierárquica, onde o nível de índice mais baixo aponta para os registros físicos e o nível mais alto aponta para o próximo nível abaixo dele.

Além disso devido ao uso de dispositivos de acesso direto (discos), o nível de índices mais baixo não precisa apontar para todos os registros de dados, mas somente para o primeiro ou o último de um bloco de registros, o que é chamada indexação parcial (Hanson, 1982).

Em resumo, para o custo de um espaço adicional de memória, de um tempo de acesso maior e de um maior esforço de manutenção, a organização sequencial indexada permite o acesso de maneira "não sequencial" por um atributo relacionado com a chave utilizada na ordenação do arquivo.

2.2.5 - ORGANIZAÇÃO ALEATÓRIA

Neste tipo de organização, os registros são armazenados e recuperados com base em uma relação determinística entre a chave do registro e o endereço da posição onde o registro está armazenado (Dodd, 1969).

Existem três maneiras para acessar os registros nos dispositivos de acesso direto:

- endereço direto,
- dicionário,
- cálculo do endereço.

O método do *endereço* direto é a forma mais simples, onde o endereço é conhecido pelo programador e fornecido na hora do armazenamento e da recuperação. O "hardware" utiliza este endereço para acessar o registro adequado.

O método do *dicionário* utiliza uma tabela, onde cada chave de acesso está armazenada juntamente com seu endereço físico. Este endereço é então utilizado para armazenar ou recuperar o regis

tro. Este método assegura que cada registro possui um endereço único, porém, para que isto possa ocorrer, o dicionário tem de ser grande o suficiente para incluir todos os possíveis endereços. É possível então que o dicionário ocupe mais espaço de memória que os próprios dados. Além disto, a procura sequencial neste caso deve ser feita passo a passo, o que a torna muito lenta.

O método do *cálculo do endereço* consiste em converter a chave do registro em um endereço direto, o qual não é necessariamente único. A conversão é feita por um algoritmo e pode ocorrer que duas chaves diferentes possuam o mesmo endereço.

O problema da geração de endereços coincidentes e conhecido como "colisão". Um dos meios de contornar este problema é utilizar uma estrutura mista, na qual os endereços que apresentam colisões são preenchidos com um ponteiro para uma lista conectada ("linked list"), a qual contém os registros que possuam o mesmo endereço. Além disto, pode ocorrer que alguns endereços nunca sejam utilizados, caso a função de cálculo não possua uma distribuição uniforme de valores.

Em resumo, a vantagem da organização aleatória é que qualquer registro pode ser recuperado com um único acesso, sem que os demais registros do arquivo precisem ser acessados e suas chaves examinadas como no método sequencial. Este tipo de organização não utiliza índices, porém faz com que o uso do espaço de armazenamento seja menos compacto que no caso da organização sequencial.

Além disso, embora a organização aleatória permita o rápido acesso a um registro qualquer com uma chave conhecida (aleatoriamente), ela torna-se lenta para o acesso a um grupo de registros de uma só vez, devido ao tempo gasto pelo mecanismo de acesso ("hardware") para localizar cada registro (Dodd, 1969).

2.2.6 - ORGANIZAÇÃO DO TIPO LISTA

O uso de ponteiros para eliminar o problema das colisões na organização aleatória sugere um outro tipo de organização de dados, chamada organização do tipo lista.

O conceito básico das listas é a utilização de ponteiros (ou apontadores), os quais isolam a organização lógica da organização física. Incluindo em cada registro um ponteiro para o próximo registro lógico, pode-se obter uma organização lógica completamente diferente da organização física, o que não ocorre na organização sequencial, onde o próximo registro lógico é também o próximo registro físico (Dodd, 1969). Existem dois tipos principais de organização do tipo lista:

- lista simples,

- lista invertida.

Na lista simples, cada registro possui um campo no qual é armazenado um ponteiro para o próximo registro lógico da lista. O último registro possui um ponteiro especial, ou nulo, que indica o fim da lista. A Figura 2.2 mostra um exemplo de lista simples.

A manutenção da lista simples é facilmente realizada. Novos registros podem ser inseridos em qualquer parte da lista, modificando o ponteiro do registro que precede o novo registro e colocando o novo ponteiro no registro que acaba de ser inserido.

A remoção de um registro da lista é feita modificando o ponteiro do registro precedente, a fim de que este aponte o registro seguinte ao que foi removido.

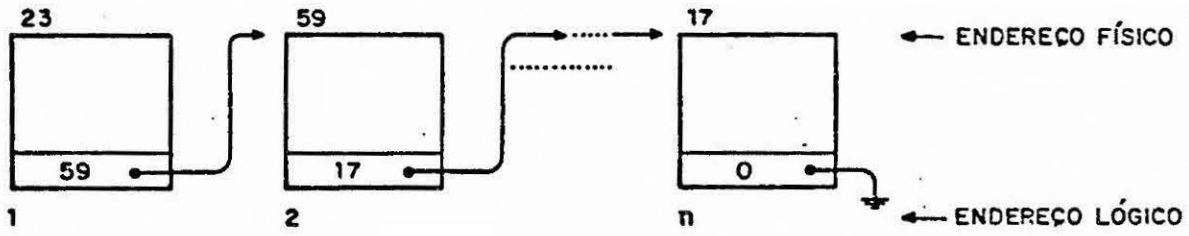


Fig. 2.2 - Lista simples.

Porém, para arquivos grandes,as próprias listas tendem a ficar longas e a procura por um dado registro torna-se muito extensa, sendo necessário o controle do tamanho da lista.

Pode-se então criar várias sublistas de acordo com várias chaves diferentes, o que irá reduzir o tempo de procura em troca de espaço para a criação de um índice dos ponteiros de início de cada sublista.

Esse tipo de estrutura é chamado *lista parcialmente invertida* e um exemplo pode ser visto na Figura 2.3.

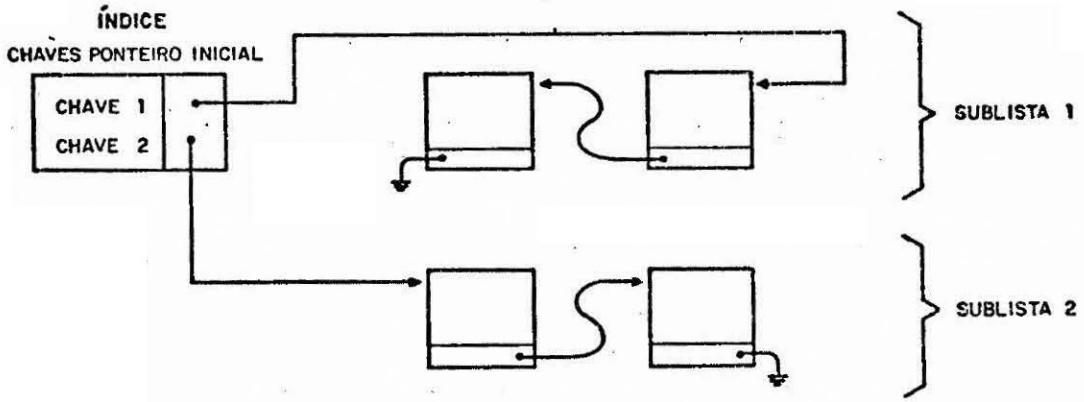


Fig. 2.3 - Lista parcialmente invertida.

Quando se tenta restringir ao máximo o comprimento da lista, esta torna-se igual a um e todas as chaves irão aparecer no índice. O índice irá então apontar diretamente para cada registro e não será necessário mais ponteiros nesta estrutura.

A lista torna-se então uma *lista invertida* e um exemplo é mostrado na Figura 2.4.

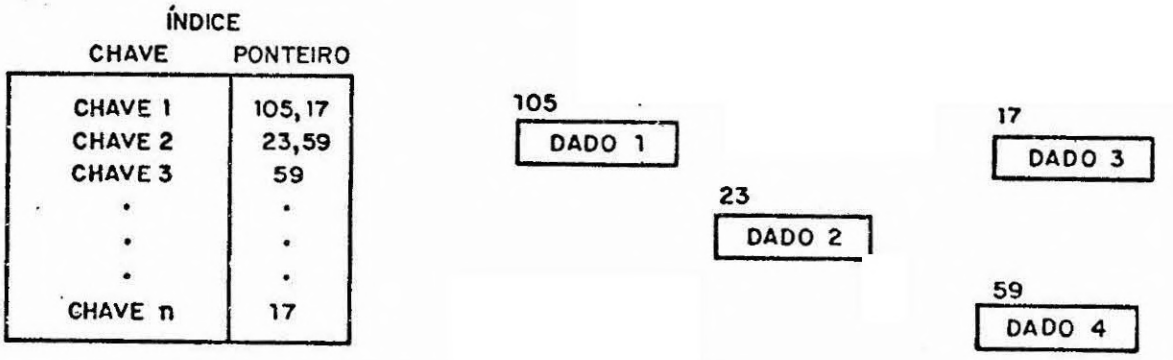


Fig. 2.4 - Lista invertida.

Mantendo os ponteiros no índice, a procura por uma determinada informação pode ser obtida apenas pelo manuseio dos ponteiros no índice e, a partir daí, tais ponteiros são usados para recuperar os dados do arquivo. É possível então processar um pedido sem consultar o arquivo propriamente dito.

Esta organização permite ainda que todos os itens de dados sejam uma chave para o acesso. Tal organização utiliza um dicionário de todos os valores dos dados contidos no sistema e de todos os endereços onde tais valores aparecem. O dicionário pode ser tão grande como, ou maior que, os dados propriamente ditos.

A *vantagem* desta organização é que ela permite o acesso a todos os dados com a mesma facilidade. A *desvantagem* é que o armazenamento e a utilização são mais difíceis devido à manutenção de grandes dicionários.

Na prática, é melhor combinar a lista invertida com outro tipo de organização como a sequencial ou aleatória. Deste modo, os registros podem ser invertidos em uma ou duas chaves ao invés de em todas elas, o que permite que os dicionários fiquem menores e ainda possam ser acessados todos os registros do arquivo (Dodd, 1969).

2.3 - SISTEMAS DE DIRETÓRIO

O diretório é utilizado basicamente para permitir a identificação e localização dos arquivos contidos em um dado meio de armazenamento. Em geral o diretório é uma lista de nomes simbólicos associados ao endereço físico ou ao "descritor" do arquivo. Algumas vezes, o diretório contém ainda outras informações sobre o arquivo, tais como: posição, comprimento, tipo, proprietário (processo ou usuário), data da criação etc.

Em sistemas de pequeno porte com um único usuário, um diretório por dispositivo é suficiente. Porém, em sistemas de maior porte, onde o espaço de armazenamento e o número de usuários é grande, torna-se mais difícil aos usuários organizar e manter o controle de todos os arquivos disponíveis. Neste caso, é necessária a imposição de uma estrutura de diretório ao sistema. Tal estrutura permite um mecanismo para a organização de todos os arquivos e pode inclusive abranger várias unidades de armazenamento ao mesmo tempo.

Desse modo, o usuário pode se preocupar somente com a estrutura lógica do diretório e dos arquivos, ignorando os problemas de alocação física dos arquivos (Peterson and Silberschatz, 1983).

Além da procura de um determinado arquivo, pode-se efetuar várias outras operações no diretório, tais como (Peterson and Silberschatz, 1983):

- criar um arquivo: novos arquivos são criados e adicionados ao sistema;

- abrir um arquivo: um arquivo já existente é colocado em uso (ativado);

- remover um arquivo: o arquivo é removido do diretório, pois não será mais utilizado;

- fechar um arquivo: o arquivo é colocado fora de uso (desativado), mas permanece no diretório;

- listar diretório: o conteúdo do diretório e, eventualmente, alguns valores das entradas dos arquivos são mostrados.

Embora o diretório tenha um papel especial no sistema e todas as operações sobre ele sejam efetuadas somente pelo Monitor de Disco, o diretório é considerado como um arquivo qualquer para o sistema. Os diretórios podem ser organizados de várias maneiras, como é o caso dos arquivos de dados. Tal organização deve facilitar a inserção, a remoção e a procura das entradas dos arquivos. Deste modo, os vários procedimentos de acesso utilizados pelo Monitor de Disco podem ser utilizados para gerenciamento do diretório (Shaw, 1974).

2.3.1 - DESCRITORES DE ARQUIVOS

A cada nome simbólico corresponde uma "entrada" no diretório, o qual associa o nome simbólico (lógico) à localização real (física) do arquivo.

Em geral, a entrada do diretório associa um nome a um "descritor do arquivo", o qual nada mais é do que uma tabela onde estão armazenadas várias informações sobre os arquivos necessários ao Sistema Operacional. Essas informações variam de sistema para sistema e podem incluir, por exemplo:

a) O nome simbólico atribuído ao arquivo.

b) O tipo do arquivo, caso o sistema suporte vários tipos.

c) Um ponteiro para uma tabela do periférico, e a posição ocupada pelo arquivo no periférico.

d) O tamanho do arquivo (em "bytes", palavras ou blocos) e o tamanho máximo permitido.

e) Um identificador único para o arquivo, o qual é o mesmo para todos os processos que usam esse arquivo (por exemplo: um número inteiro).

f) O tipo de acesso permitido (leitura, escrita, execução, etc) e os processos aos quais é permitido o acesso.

g) O número de vezes que o arquivo foi acessado, ou os processos que estão utilizando o arquivo.

h) As datas que o arquivo foi criado, a data em que foi feita a última modificação e a data em que foi feito o último acesso.

Tais informações irão permitir, entre outras coisas, que o Monitor de Disco faça o mapeamento dos registros lógicos para os registros físicos.

Um descritor é montado quando um arquivo é criado pela primeira vez e atualizado, quando o arquivo é movido, contraído, expandido ou acessado (Shaw, 1974). O armazenamento de tais informações pode utilizar uma quantidade de memória que varia de 16 até mais de

1000 "bytes" por arquivo. Em sistemas com um grande número de arquivos, o tamanho do diretório pode ser da ordem de vários milhares de "bytes", tornando-se necessário que o diretório seja mantido no próprio disco e trazido em partes para a memória conforme haja necessidade, isto é, quando o arquivo for ativado (Peterson and Silberschatz, (1983).

Na prática, os descritores são sempre armazenados em uma área separada do disco que contém os arquivos correspondentes. Deste modo, possibilita-se o uso de descritores maiores em tamanho, fazendo com que o conteúdo de um periférico seja identificado dentro do próprio periférico. Além disto, este método permite que várias entradas do diretório possam apontar para um mesmo arquivo.

Normalmente, o descritor é mantido na memória principal desde o instante em que o arquivo se torna ativo (aberto) até o instante em que ele seja desativado (fechado); durante esse tempo, o descritor deve ser atualizado pelas diversas rotinas de processamento do Monitor de Discos (Shaw, 1974).

2.3.2 - DIRETÓRIO DE UM NÍVEL

A estrutura mais simples é a do diretório de um único nível, onde todos os arquivos estão contidos no mesmo diretório, o qual é simples de ser trabalhado e mantido. A Figura 2.5 mostra esta estrutura (Peterson and Silberschatz, 1983).

O diretório de um nível, no entanto, possui limitações significativas quando o número de arquivos é aumentado, ou quando existem vários usuários ou processos no sistema. Desde que todos os arquivos estejam no mesmo diretório, eles devem possuir nomes diferentes, ou seja, o usuário não pode atribuir nomes já existentes, ainda que tais nomes sejam convenientes a ele. Mesmo no caso de um único usuário possuir um grande número de arquivos, torna-se difícil lembrar os nomes de todos os arquivos já existentes, de modo a não criar novos arquivos com nomes já atribuídos a outros arquivos.

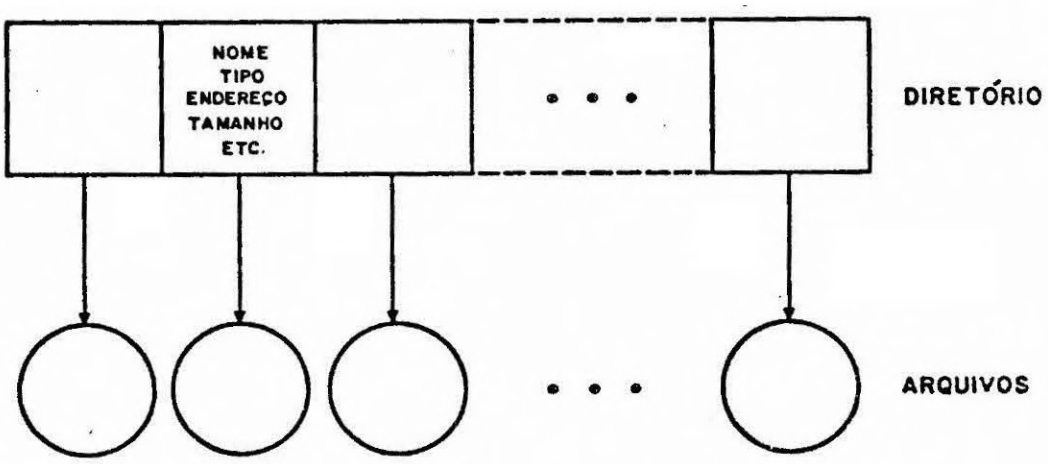


Fig. 2.5 - Diretório de um nível.

2.3.3 - DIRETÓRIO DE DOIS NÍVEIS

A maior desvantagem do diretório de um nível é o problema da atribuição de nomes idênticos a dois ou mais arquivos. A solução mais comum é a criação de diretórios "separados" para cada usuário. Tal separação, no entanto, pode ser apenas lógica e não física, desde que todos os arquivos possam, contudo, estar armazenados num mesmo dispositivo físico.

Cada processo ou usuário possui seu próprio diretório e cada diretório tem uma estrutura similar, mas que contém somente os arquivos de um único processo.

Existe um diretório mestre de arquivos que contém uma lista de todos os nomes dos processos do sistema num dado instante. Quando um processo é ativado, seu nome é procurado no diretório mestre, o qual é indexado por esse nome e cuja entrada aponta para o diretório desse processo, como pode ser visto na Figura 2.6 (Peterson and Silberschatz, 1983).

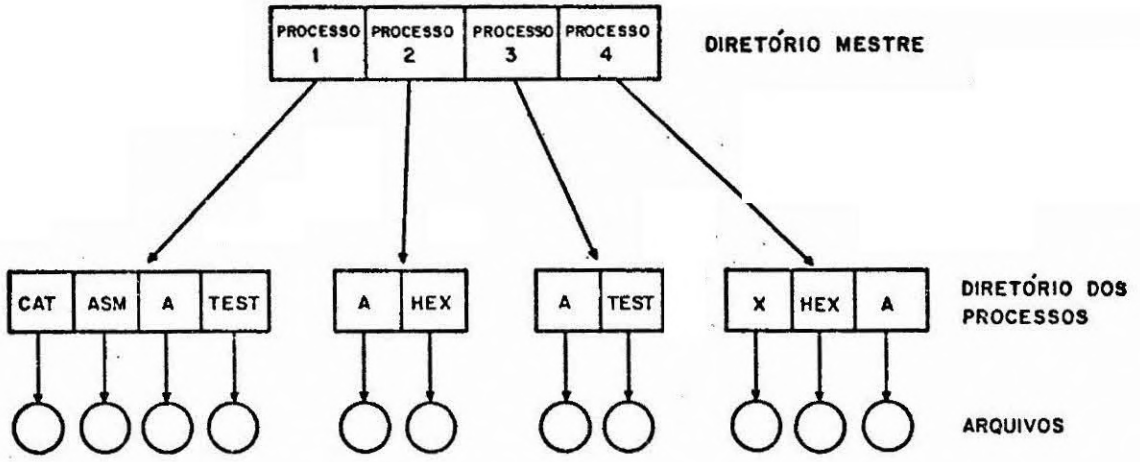


Fig. 2.6 - Diretório de dois níveis.

Nessa estrutura, quando é feita a referência a um determinado arquivo, o Monitor de Disco procura somente no diretório do processo requisitante. Neste caso, processos diferentes podem possuir arquivos com um mesmo nome. Na criação e remoção de arquivos, a procura é feita apenas no diretório de um processo; isto evita que um arquivo com o mesmo nome, porém de outro processo seja removido.

O problema existente com o diretório de dois níveis é que tal estrutura isola os processos entre si. Isto pode ser uma vantagem quando os processos são independentes, mas é uma desvantagem quando existe a necessidade de cooperação entre processos que desejam compartilhar arquivos comuns. No caso onde o compartilhamento é necessário, deve haver um modo de um processo referenciar-se a um arquivo de outro processo.

Pode-se considerar um diretório de dois níveis como se fosse uma estrutura do tipo árvore de altura dois, cuja raiz é o diretório mestre e os descendentes diretos são os diretórios dos proces

sos. Os descendentes dos diretórios de processos são os próprios arquivos que também são as folhas da árvore.

A especificação de um nome de um processo e de um nome de arquivo define um "caminho" na árvore a partir da raiz até uma folha (o arquivo especificado). Portanto, um nome de processo e um nome de arquivo definem um nome de caminho ("path name"), ou simplesmente caminho. Todo arquivo no sistema possui um único caminho. Para compartilhar um arquivo deve-se especificar o nome do caminho do arquivo desejado (Peterson and Silberschatz, 1983).

2.3.4 - DIRETÓRIO DO TIPO ÁRVORE

A generalização do diretório de dois níveis é uma árvore de tamanho arbitrário. Tal estrutura permite a um processo ou usuário criar subdiretórios para organizar seus próprios arquivos. A Figura 2.7 mostra uma estrutura do tipo árvore (Peterson and Silberschatz, 1983).

Nessa estrutura, cada nó da árvore é um arquivo diretório e cada ramo é uma entrada do diretório que aponta para outro diretório (subdiretório) ou para um arquivo de dados. Os arquivos de dados propriamente ditos são as folhas da árvore. A raiz da árvore é chamada diretório "mestre" ou diretório "raiz".

Todos os arquivos possuem um caminho com nome único ("path name") formado pelo nome dos ramos desde a raiz, passando por todos os subdiretórios, até o arquivo especificado. Até o momento, considerou-se que cada arquivo é referenciado por um único ramo do diretório. Caso contrário, o nome simbólico não será mais único, isto é, poderá existir mais de um caminho partindo da raiz até o arquivo. Mesmo assim, cada caminho ainda define um único arquivo.

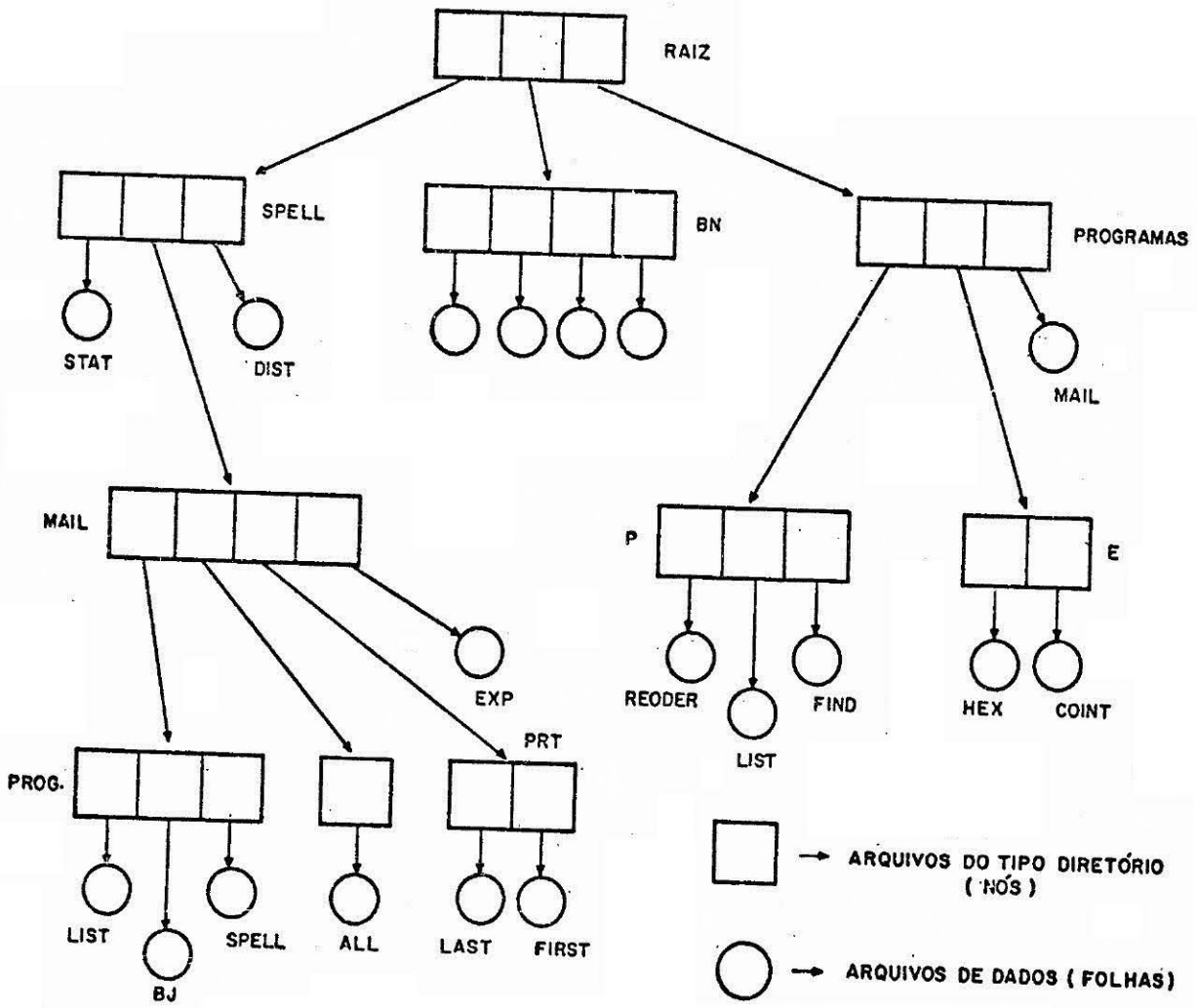


Fig. 2.7 - Diretório do tipo árvore.

Esse esquema de identificação através dos nomes dos caminhos evita os conflitos de identificação entre os vários processos e praticamente permite qualquer atribuição arbitrária de nomes para os arquivos. O nome do caminho é interpretado como o nome dado ao arquivo pelo processo, enquanto ele está trabalhando dentro de um dado contexto. O nome do caminho pode ser empregado como um argumento de busca para encontrar qualquer arquivo de diretório ou de dados.

A grande vantagem da estrutura em árvore é que os arquivos podem ser facilmente compartilhados, isto é, os arquivos criados por um processo podem ser acessados por outros processos, desde que seja especificado o nome do caminho para tais arquivos. Por outro lado, os mecanismos de controle de acesso e proteção (quando desejados) tem de ser implementados de maneira mais eficaz nesta estrutura.

2.3.5 - DIRETÓRIO DO TIPO GRAFO ACÍLICO

A estrutura do tipo árvore não permite o compartilhamento total de arquivos ou diretórios. A estrutura do tipo grafo acíclico, que é uma generalização da estrutura em árvore, permite que os diretórios possuam arquivos e subdiretórios compartilhados. O grafo acíclico não possui ciclos na sua estrutura, como mostra a Figura 2.8 (Peterson and Silberschatz, 1983).

Com o grafo acíclico o usuário pode definir seus próprios subdiretórios e assim impor uma estrutura aos arquivos. Esta estrutura pode ser formada por diretórios separados para arquivos associados com diferentes tópicos ou diferentes formas de informação. Como exemplo, pode-se ter um diretório chamado "fonte" com programas fontes e o diretório "binário" com os códigos objeto.

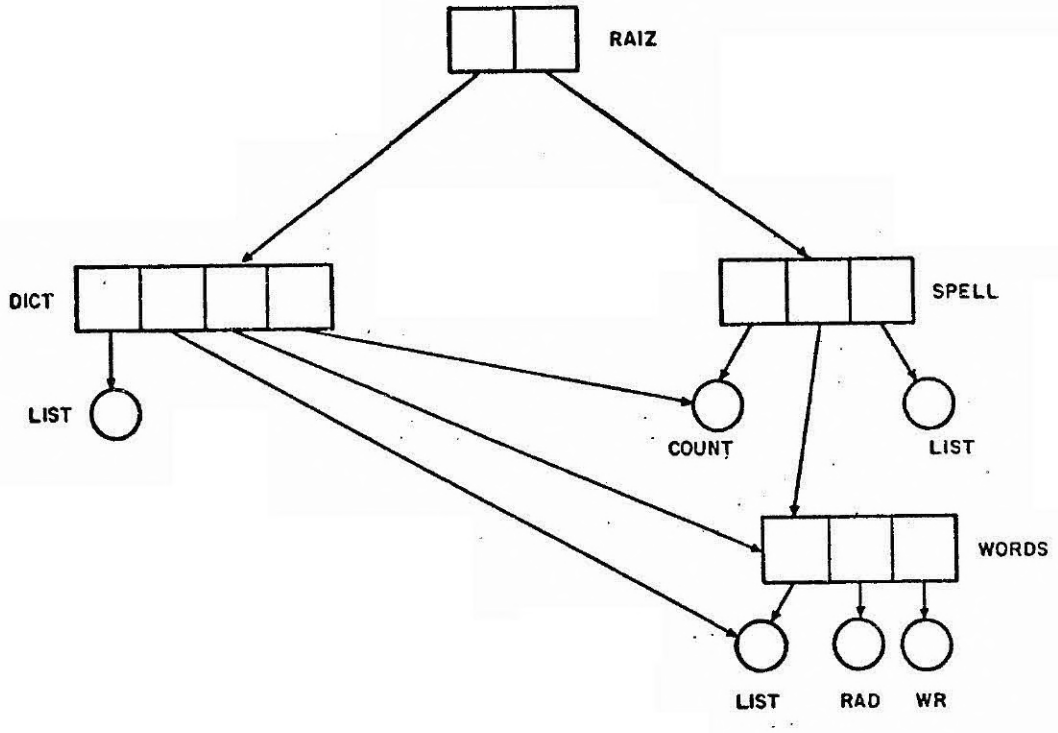


Fig. 2.8 - Diretório do tipo grafo acíclico.

A estrutura de diretório do tipo grafo acíclico é mais flexível que a do tipo árvore, porém é mais complexa e vários problemas devem ser considerados. Nesta estrutura, um arquivo pode ter vários caminhos para ser acessado e, portanto, nomes diferentes podem referir-se a um mesmo arquivo. Caso seja necessário consultar todo o sistema de arquivos (para encontrar um arquivo, copiar todos os arquivos etc.), o problema de existirem vários caminhos pode ser crítico, pois não é desejável que o algoritmo de procura passe por caminhos compartilhados mais de uma vez.

Outro problema é o da remoção de arquivos, isto é, como determinar quando o espaço alocado a um arquivo compartilhado pode ser devolvido ao sistema e reutilizado. Uma alternativa é remover o arquivo sempre que alguém o deseje, mas isto pode fazer com que os ponteiros vindos de outros diretórios permaneçam apontando para um arquivo que já não existe. Pior seria se as referências pendentes do arquivo removido fossem endereços físicos do disco e o espaço deste arquivo fosse posteriormente reutilizado para outros arquivos. Os ponteiros pendentes poderiam apontar para o meio dos outros arquivos. Em sistemas onde o compartilhamento é implementado através de conexões ("links"), o problema da remoção é mais facilmente resolvido, pois a remoção de uma conexão não afeta o arquivo original, mas somente a conexão.

Se a entrada para um arquivo é removida, o espaço para o arquivo é liberado e as conexões ficam pendentes. Pode-se então localizar tais conexões e removê-las; porém, se não existir uma lista das conexões associadas a cada arquivo, tal procura torna-se muito dispendiosa.

Para resolver o problema da remoção, pode-se preservar o arquivo até que todas as referências a ele tenham sido removidas. Para implementar isto, deve-se encontrar um meio de determinar se a última referência ao arquivo já foi removida.

Pode-se manter uma lista de todas as referências a um arquivo (entradas de diretório ou conexões), de modo que sempre que uma conexão é estabelecida, um novo item é adicionado à lista de referências e, vice-versa; quando uma conexão ou entrada do diretório é removida, remove-se sua entrada na lista. O arquivo é removido quando sua lista de referência estiver vazia.

O problema com essa alternativa é o tamanho variável e potencialmente grande da lista de referências a um arquivo.

Entretanto, não há necessidade de manter a lista em si, mas somente a contagem do número de referências ao arquivo. Uma nova conexão ou entrada de diretório incrementa a contagem, e a remoção de uma conexão ou entrada de diretório decrementa a contagem. Quando a contagem for igual a zero, o arquivo pode ser removido, pois não existem mais referências a ele (Peterson and Silberschatz, 1983).

2.3.6 - DIRETÓRIO DO TIPO GRAFO SIMPLES

A simples adição de novos arquivos e subdiretórios a um diretório estruturado em árvore não afeta a estrutura deste. Porém, ao serem adicionadas conexões ("links") a um diretório do tipo árvore, a estrutura da árvore é destruída, resultando numa estrutura do tipo grafo simples, como mostrado na Figura 2.9 (Peterson and Silberschatz, 1983).

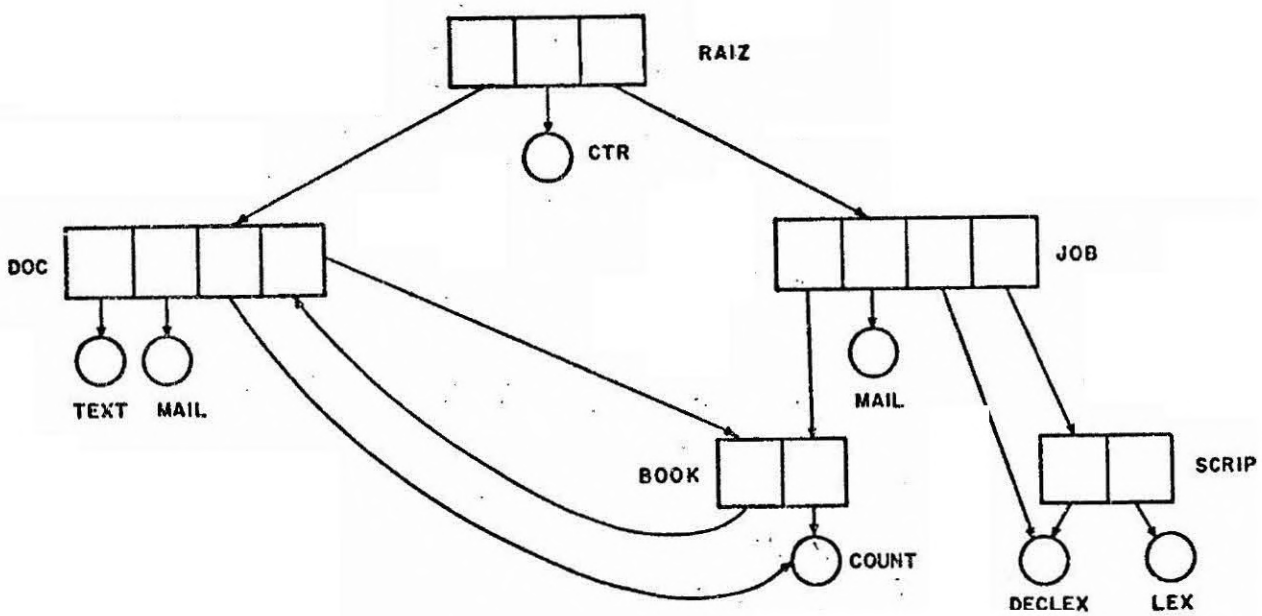


Fig. 2.9 - Diretório do tipo grafo simples.

Ao percorrer uma estrutura do diretório que possua ciclos, deseja-se evitar que um mesmo componente seja procurado mais de uma vez por razões de integridades e desempenho. Um algoritmo mal concebido poderia entrar num ciclo de procura infinita e nunca terminar.

Existe ainda o problema de determinar quando é que um arquivo pode ser removido. Como no caso citado para o grafo acíclico, quando uma variável de contagem do número de referências ao arquivo ou diretório for zero, isto indica que não existem mais referências a ele, podendo ser removido. Entretanto, quando existirem ciclos, é possível que o valor da contagem de referência seja não-nulo, mesmo quando já não seja mais possível fazer referência a um arquivo ou diretório. Este problema resulta da possibilidade do algoritmo se auto-referenciar (através dos ciclos) na estrutura do diretório.

Nesse caso é necessário utilizar algoritmos de limpeza ("garbage collection") para determinar quando a última referência a um arquivo ou diretório foi removida e quando o espaço pode ser devolvido e reutilizado.

Essa limpeza é feita percorrendo toda a estrutura de arquivos e marcando tudo o que pode ser acessado; num segundo plano, coloca-se tudo o que não foi marcado na lista de espaço livre. Entretanto, o algoritmo de limpeza para sistemas com discos consome muito tempo para ser executado e raramente é realizado.

Essa limpeza é necessária somente devido aos possíveis ciclos no grafo. Logo, uma estrutura do tipo grafo acíclico é mais fácil de ser utilizada. A dificuldade é evitar os ciclos, assim que vão sendo adicionadas novas conexões ("links") à estrutura. Existem algoritmos para localizar ciclos em grafos, porém eles consomem muito tempo, especialmente quando se utiliza armazenamento em discos.

2.4 - OPERAÇÕES EM ARQUIVOS

Um arquivo é um tipo abstrato de dados e para operá-lo corretamente deve-se definir quais as várias "operações" que podem ser realizadas com ele; tais operações são invocadas através das primitivas disponíveis e irão gerar comandos para os dispositivos físicos de E/S como mostra a Figura 2.10.

As primitivas são o meio pelo qual se pode acessar os arquivos do sistema; este acesso pode ser feito de forma sequencial ou direta, independente da organização adotada para o arquivo.

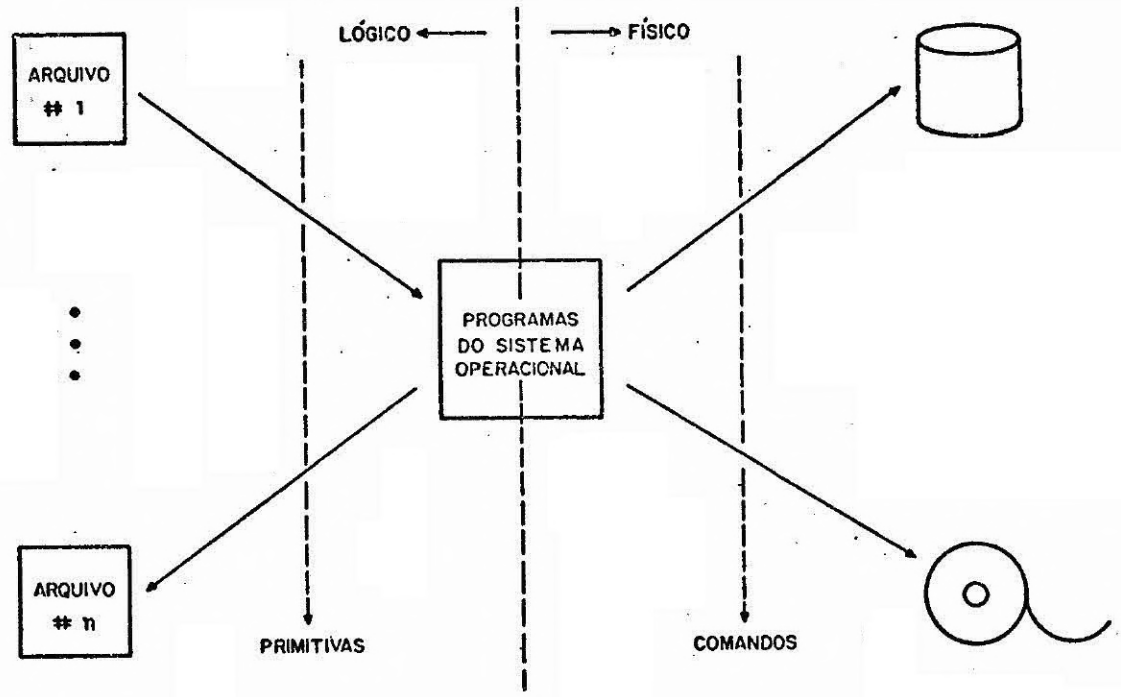


Fig. 2.10 - Mapeamento dos arquivos.

2.4.1 - PRIMITIVAS

Em um determinado sistema, vários usuários podem estar operando com vários arquivos simultaneamente e durante longos períodos de tempo. Como, em geral, o número total de arquivos do sistema pode ser grande, ocorre que é mais eficiente manter na memória principal somente as informações sobre os arquivos em uso naquele período de tempo. Quando o arquivo não for mais necessário, ele deve ser removido ou colocado fora de uso.

Torna-se clara então a necessidade do Monitor de Disco possuir procedimentos para ativar ("abrir") e desativar ("fechar") os arquivos do sistema. Tais procedimentos são realizados pelas operações de abertura ("open") e fechamento ("close") de arquivos.

Pode-se exigir que todo arquivo seja aberto antes de que qualquer outra operação (exemplo: leitura, escrita, modificação) seja nele efetuada, ou então, o sistema abre o arquivo automaticamente quando ele for referenciado pela primeira vez. Da mesma maneira, pode-se ter de fechar explicitamente um arquivo, ou o sistema automaticamente fechá-lo quando o processo que o estava utilizando terminar.

Todas as operações efetuadas sobre os arquivos são realizadas por primitivas do Monitor de Discos, implementando a interface entre o "hardware" da máquina e o processo aplicativo.

Tais rotinas formam uma estrutura lógica que será utilizada pelos programas de aplicação para acesso às informações contidas nos dispositivos físicos de armazenamento. Portanto, essas primitivas são responsáveis pela interface entre o Monitor de Discos e o Sistema Operacional da máquina utilizada.

Atualmente existem vários Sistemas Operacionais em uso que, em sua maioria, não são compatíveis entre si, tornando difícil o desenvolvimento de programas portáteis em áreas que envolvam o Siste

ma Operacional, como é o caso dos gerenciadores de arquivos. Uma maneira de minimizar o problema da portabilidade é utilizar uma interface virtual e padronizada, a qual seria então suportada por vários Sistemas Operacionais em múltiplos ambientes de "hardware".

A proposta do padrão MOSI - "Microprocessor Operating Systems Interface" do IEEE (1984) apresenta um conjunto padronizado de primitivas, as quais abrangem as várias áreas do sistema operacional.

As primitivas do grupo de transferência de dados ("data transfer group") e do módulo de E/S síncrona ("synchronous I/O module") são um exemplo típico das primitivas que devem ser implementadas pelo Monitor de Disco.

As primitivas do monitor proposto são elaboradas com base nas existentes no padrão MOSI. O monitor deve também possuir um conjunto de primitivas utilizadas para sua configuração inicial.

A Tabela 2.1 apresenta o conjunto de primitivas do padrão MOSI acima citadas.

2.5 - PROTEÇÃO DE ARQUIVOS

Um dos tópicos mais importantes num sistema de informação é a proteção dessas informações, tanto do ponto de vista dos danos físicos (confiabilidade) quanto ao acesso indevido (proteção).

A confiabilidade geralmente é conseguida através da duplicação dos arquivos do sistema, o que pode ser feito automaticamente pelo Sistema Operacional em intervalos regulares, dependendo de cada aplicação. A cópia é feita em fitas magnéticas e mantida para o caso de destruição acidental dos arquivos em disco.

TABELA 2.1

PRIMITIVAS DO PADRÃO MOSI

"DATA TRANSFER GROUP" "MODULE 2" "SYNCHRONOUS FILE I/O"
CONECT
DISCONNECT
CREATE
EXTEND
TRUNCATE
OPEN
CLOSE
READ
WRITE
SEEK
GET-FILE-INFORMATION
GET-ACCESS-CONTROL
CHANGE-ACCESS-CONTROL
DELETE
RENAME
GET-WORKING-DIRECTORY
CHANGE-WORKING-DIRECTORY
GET-IDENTITY

A proteção pode ser conseguida de várias formas. Para um sistema monousuário, a proteção pode ser implementada, por exemplo, instruindo cada usuário a remover fisicamente seu disco após o término de uma sessão. Entretanto, em sistemas multiusuários de maior porte existe a necessidade de outros mecanismos de proteção, devido à possibilidade do compartilhamento de arquivos existentes nestes sistemas (Peterson and Silberschatz, 1983).

Existem vários mecanismos de proteção que permitem o compartilhamento controlado de arquivos, limitando o tipo de acesso que pode ser feito a ele. O acesso é permitido ou negado dependendo de vários fatores, um dos quais é o tipo de acesso desejado.

Os vários sistemas existentes permitem o controle de várias operacões nos arquivos, dentre as quais podem ser citadas:

- Leitura: permissão para ler o arquivo.

- Escrita: permissão para escrever ou trancar o arquivo.

- Extensão: permissão para adicionar informação ao arquivo.

- Execução: permissão para executar o código do arquivo.

- Remoção: permissão para remover o arquivo.

- Modificação do Acesso: permissão para modificar o tipo de acesso ao arquivo.

- Acesso Exclusivo: o usuário tem uso exclusivo enquanto estiver usando o arquivo.

Os mecanismos de proteção são implementados nas operações de nível mais baixo; entretanto, a proteção em níveis mais altos, tais como permissão para mudar o nome de um arquivo, copiar ou

editar um arquivo pode ser conseguida quando se considerar que a cópia ou edição pode ser realizada como uma simples sequência de operações de leitura do arquivo.

Todos os mecanismos de proteção acima citados podem ser implementados com base em tabelas que informam "quem" pode acessar e "como" se pode acessar um arquivo contidas no descritor do arquivo.

Existe um outro tipo de acesso que não está representado nas informações de controle de acesso do descritor do arquivo, mas que precisa ser controlado: é o caso em que vários processos podem, potencialmente, acessar um determinado arquivo simultaneamente.

Não surgem problemas quando vários processos lêem um mesmo arquivo ao mesmo tempo. Entretanto, se um processo está escrevendo em um determinado arquivo, nenhum outro acesso (leitura ou escrita) deve ser permitido neste arquivo durante o tempo de escrita, senão, os resultados serão imprevisíveis (Shaw, 1974).

Esse tipo de problema pode ser resolvido com o uso de ferramentas de "software" do tipo semáforo, pois é uma variação do conhecido problema de regiões críticas.

Na implementação do sistema de arquivos "Mesa" (Guarino Reid and Karlton, 1983), é permitido um sofisticado compartilhamento de arquivos entre processos independentes. Por exemplo, se um processo precisa escrever num arquivo que está sendo lido por outro processo, é solicitado a este que pare a leitura. Pode ocorrer ainda que um processo peça para ser notificado quando um arquivo se torna disponível para um determinado tipo de acesso.

O sistema "Mesa" facilita a cooperação entre os processos, permitindo que estes possuam procedimentos que o sistema de arquivos pode chamar para pedir que um processo entregue um arquivo, ou para notificar um processo que um determinado arquivo está disponível.

As operações que devem ser protegidas nos arquivos do tipo diretório são um pouco diferentes, pois neste caso, deseja-se controlar a criação ou remoção dos arquivos num diretório. Há possibilidade de controlar também se um usuário pode determinar a existência de um arquivo no diretório, caso a listagem do conteúdo de um diretório seja uma operação protegida. Neste caso, o conhecimento da existência e do nome de um arquivo pode ser significativo (Peterson and Silberschatz, 1983).

O sistema "UNIX" possui rotinas para manuseio de diretórios invocáveis somente pelo chamado "super-user" (superusuário), o qual pode criar um diretório. Neste caso, todos os usuários possuem uma identificação ("id") e o sistema reconhece um usuário particular ("super-user") que é dispensado dos controles de acesso normais existentes para os demais usuários (Ritchie and Thompson, 1974).

Existem vários mecanismos de proteção disponíveis atualmente, cada qual com suas vantagens e desvantagens, e a escolha do melhor mecanismo deve ser feita de acordo com a aplicação desejada. Alguns desses mecanismos são apresentados a seguir.

2.5.1 - CONTROLE DO ACESSO POR NOME

Um dos esquemas mais simples de proteção, existentes em muitos sistemas, depende do fato de que um usuário não pode acessar um arquivo se ele não conhece o nome deste. Se o usuário não conhece o nome, então ele não pode operar sobre o arquivo. Este sistema considera que não existem meios de obter os nomes dos arquivos dos usuários e por isto não podem ser facilmente encontrados. Tal esquema tem segurança limitada, pois, como geralmente os nomes de arquivos tendem a ser mnemônicos, eles sempre podem ser facilmente descobertos.

2.5.2 - CONTROLE DO ACESSO POR SENHA

Outra alternativa é associar uma senha ("password") a cada arquivo. Da mesma maneira que o acesso ao próprio sistema de computação é controlado por uma senha, o acesso a cada arquivo também pode ser controlado por ela. Caso a senha seja escolhida aleatoriamente e modificada sempre, esse esquema torna-se bastante efetivo, limitando o acesso a um arquivo somente a pessoas que conhecem a senha.

Normalmente, no entanto, só uma senha é associada a cada arquivo, e sua proteção torna-se do tipo tudo ou nada. Para permitir esta proteção num nível mais detalhado, é necessário o uso de múltiplas senhas.

2.5.3 - CONTROLE DO TIPO DE ACESSO

Essa alternativa faz o acesso ser dependente da identidade do processo. Vários processos podem desejar diferentes tipos de acesso a um arquivo ou diretório. Pode então ser utilizada uma lista de acesso associada a cada arquivo ou diretório, especificando o nome do processo e o tipo de acesso permitido a cada um. Quando um processo pede o acesso a um dado arquivo, é verificada a lista de acesso do arquivo e, caso o processo tenha direito, o acesso é permitido. Caso contrário, ocorre um erro de violação da proteção do arquivo e o trabalho do processo é abortado.

O maior problema das listas de acesso é o seu tamanho, que pode ser muito grande se existir, por exemplo, um arquivo que possa ser lido por todos. A lista de acesso deste arquivo deveria possuir todos os nomes dos processos. Utiliza-se então uma maneira resumida de lista de acesso, na qual o sistema reconhece três tipos de processos: o criador do arquivo, um grupo de processos e um processo qualquer. Cada arquivo possui o nome de seu criador e ele próprio pode pertencer a um grupo de processos que estão compartilhando do mesmo arquivo e precisam do mesmo tipo de acesso. Os membros de uma classe ou departamento podem definir um grupo e existe ainda o processo qualquer.

Nessa classificação são precisos apenas três campos para definir a proteção. Por exemplo, o sistema "UNIX" define três campos de três bits cada um: "rwx", onde "r" controla o acesso para leitura, "w" controla o acesso para escrita e "x" controla a execução. É mantido um campo separado para o nome do criador, para o grupo ao qual pertence o nome do criador e outro para todos os usuários. Tal esquema utiliza 9 bits para armazenar as informações de proteção (Ritchie and Thompson, 1974).

A proteção pode ser associada com o arquivo propriamente dito, ou com o nome do caminho usado para especificar o arquivo. A maneira mais comum é incluir a proteção no nome do caminho. Em sistemas onde um arquivo pode ser acessado por vários caminhos (no caso de grafos simples ou acíclicos), cada processo deve possuir diferentes direitos de acesso a um arquivo, dependendo de qual caminho é especificado.

2.6 - ALOCAÇÃO DE ESPAÇO DA MEMÓRIA SECUNDÁRIA

Em sistemas baseados em discos magnéticos existe uma grande flexibilidade na implementação de arquivos. O espaço disponível é geralmente compartilhado por muitos arquivos e o maior problema é como manter o controle do espaço da memória secundária, ou seja, como alocar o espaço aos vários arquivos, de tal modo que esse espaço seja efetivamente utilizado e os arquivos possam ainda ser facilmente acessados. Como é natural, algumas das técnicas utilizadas para o gerenciamento da memória principal são aplicáveis aqui, porém não se pode esquecer das modificações necessárias para o caso da memória secundária.

A função básica do alocador é atender aos pedidos da memória secundária, que podem ser "blocos" de tamanho fixo ou variável. Um bloco é um conjunto de setores do disco. Geralmente o espaço é alocado em unidades de blocos de tamanho fixo, devido à organização físi

ca dos meios de armazenamento ser feita em setores de tamanho fixo, como é o caso dos discos magnéticos. Isto se deve também à grande quantidade de espaço disponível nos discos e à simplicidade de tal política.

A eficiência da E/S é o principal critério a ser utilizado para decidir quais blocos livres devem ser selecionados, pois deseja-se minimizar os tempos de busca e atrasos rotacionais quando os blocos forem posteriormente acessados (Shaw, 1974). Os três métodos mais usados atualmente para a alocação do espaço da memória secundária, aqui analisados, são: alocação contínua, alocação conectada e alocação indexada.

2.6.1 - GERENCIAMENTO DO ESPAÇO LIVRE

Durante a operação normal de um sistema de computação, os arquivos são criados e removidos de maneira bastante irregular. Desde que o espaço disponível na memória secundária é limitado, é necessário que o espaço proveniente da remoção de arquivos seja reutilizado para a criação de novos arquivos.

Existem vários métodos usados para manter o controle do espaço livre da memória secundária, sendo que todos envolvem a existência de uma estrutura de dados conhecida como "lista de espaço livre", a qual contém informações sobre todos os registros que estão livres, isto é, não-alocados em nenhum arquivo. Para a criação de um arquivo, procura-se na lista de espaço livre o espaço necessário e aloca-se este espaço para o novo arquivo. Quando um arquivo é removido, o espaço anteriormente ocupado por ele é adicionado à lista de espaço livre.

De acordo com Peterson e Silberschatz (1983), a maneira mais comum de estruturar uma lista de espaço livre é ligar todos os blocos livres através de ponteiros contidos em cada um deles, mantendo na memória principal apenas um ponteiro para o primeiro bloco li

vre. Esta maneira tem a vantagem de ser simples, porém é muito inefi ciente. Para percorrer a lista durante a adição ou remoção de blocos, é necessária a leitura de todos os setores e, posteriormente, a modi ficação de ponteiros, o que consome muito tempo em operações de E/S. Uma modificação deste método é armazenar os endereços dos primeiros "n" blocos livres no primeiro bloco, dos quais n-1 estão realmente li vres. A vantagem desse método é que os endereços de um grande número de blocos livres podem ser encontrados rapidamente, de uma única vez.

Devido a problemas da lista conectada por ponteiros, na prática é mais empregada a técnica de tabelas de índices, semelhante àquela citada anteriormente como uma das maneiras de organização físi ca de arquivos. Esta técnica utiliza uma tabela ou conjunto de tabe las por dispositivo de armazenamento, as quais contêm todo o espaço livre disponível. Tais tabelas podem ser organizadas internamente co mo listas conectadas ou como "vetores de bits".

No primeiro caso, é alocada uma palavra para cada bloco livre da memória secundária e todos os blocos livres são ligados jun tos por ponteiros. Deste modo é relativamente fácil alocar e desalo car blocos, porém há um grande consumo de espaço de memória para man ter tal estrutura.

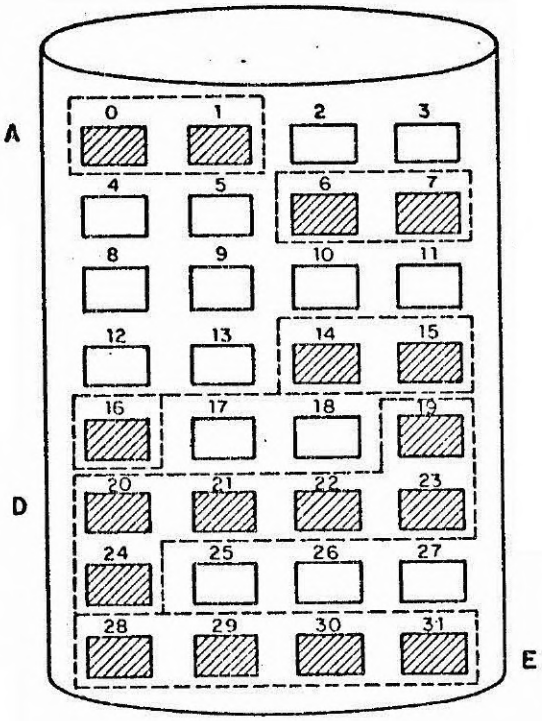
No segundo caso, a economia de espaço é bem maior, pois cada bloco é representado por apenas um bit na tabela. Por exemplo, se o bit for igual a 1, o bloco estará alocado e se for igual a 0, o blo co estará livre. A liberação de blocos livres e a representação mais eficiente tornam geralmente preferível esta técnica.

2.6.2 - ALOCAÇÃO CONTÍNUA

A alocação contínua é o método mais simples, no qual ca da arquivo ocupa um conjunto de endereços contínuos no disco. Os ende reços do disco definem uma ordenação linear no disco. Normalmente os endereços aumentam nos setores de uma trilha, depois nas trilhas de

um cilindro e finalmente até todos os cilindros serem utilizados. Deste modo, para acessar o setor S+1, estando no setor S, em geral é desnecessário um movimento da cabeça do disco.

A alocação contínua de um arquivo é definida apenas pelo endereço do primeiro setor e pelo comprimento do arquivo. Se um arquivo tem "n" setores e inicia no setor "S", então ele ocupa os arquivos S, S+1, S+2,..., S+n-1. A entrada no diretório para cada arquivo precisa conter apenas o endereço do setor inicial e o número de setores alocados para o arquivo, como pode ser visto na Figura 2.11 (Peterson and Silberschatz, 1983).



DIRETÓRIO:

ARQUIVO	INÍCIO	COMPRIMENTO
A	Ø	2
B	6	2
C	14	3
D	19	6
E	28	4

Fig. 2.11 - Alocação contínua.

O acesso a um arquivo alocado continuamente é bastante simplificado. No caso de acesso sequencial, a rotina de acesso precisa somente do endereço do último setor acessado para acessar o próximo. No caso de acesso direto a um determinado bloco "i" de um arquivo que inicia no setor "S", pode-se acessar imediatamente o setor S+1. Portanto, a alocação contínua suporta o acesso direto e o sequencial.

A maior dificuldade com a alocação está em como encontrar espaço para a alocação de um arquivo, a partir da lista de espaço livre. Se o arquivo a ser alocado tem o comprimento de "n" setores livres contínuos, é preciso encontrar na lista de espaço livre "n" setores livres contínuos. Este problema pode ser visto como um caso particular do problema da alocação dinâmica de memória, o qual resolve o como satisfazer um pedido de tamanho "n" a partir de uma lista de espaço livre (Knuth, 1973).

As soluções mais comuns são a procura do tipo "first-fit" ou "best-fit". Tais soluções consistem em pesquisar a lista de espaço livre até encontrar o primeiro espaço de tamanho suficiente para o arquivo ("first-fit"), ou até encontrar o menor espaço livre de tamanho suficiente para o arquivo ("best-fit"). Estudos sobre a eficiência de tais algoritmos (Shore, 1975) não chegaram a uma conclusão sobre qual é o melhor, porém o método "first-fit" é pelo menos mais rápido.

Ambos os métodos acima causam a fragmentação externa. Conforme os arquivos vão sendo alocados e removidos, o espaço livre vai sendo dividido em muitos espaços pequenos, chegando a um ponto em que existirá espaço livre disponível para um arquivo, porém este espaço não será contínuo.

Outro problema com a alocação contínua é a determinação de quanto espaço será necessário para armazenar um arquivo. Este espaço precisa ser especificado na hora da criação do arquivo, o que em muitos casos é muito difícil de ser estimado. Caso seja alocado pouco

espaço para o arquivo, este espaço não pode ser estendido. Pode-se então abortar o programa, enviar uma mensagem ao processo que deverá alocar mais espaço para o arquivo e rodar novamente o programa. Este método é muito trabalhoso e acaba levando a um superestimação da necessidade de espaço, o que causa uma perda desnecessária de espaço livre (Peterson and Silberschatz, 1983).

2.6.3 - ALOCAÇÃO POR LISTA CONECTADA

Os maiores problemas da alocação contínua decorrem do fato de ela exigir que o espaço do disco seja alocado continuamente. A alocação conectada não apresenta este problema, pois cada arquivo é formado por uma lista conectada de vários setores do disco. O diretório contém um ponteiro para o primeiro (e talvez também para o último) setor do arquivo. Por exemplo, um arquivo com 5 setores pode iniciar no setor 9 e continuar no setor 16, depois no setor 1, no setor 10 e finalmente no setor 25, como na Figura 2.12. Cada setor possui um ponteiro para o próximo setor.

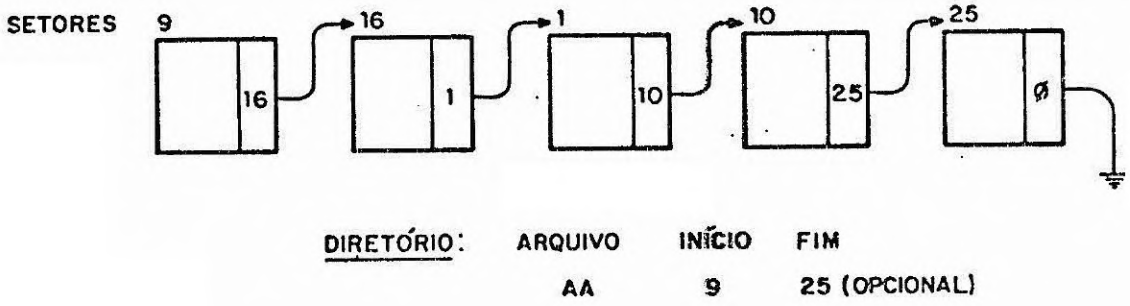


Fig. 2.12 - Alocação por lista conectada.

A criação de um arquivo é simples: adiciona-se uma nova entrada no diretório, isto é, um ponteiro para o primeiro setor do arquivo, o qual é inicializado como um valor nulo ("nil"), o que in

dica que o arquivo está vazio. Uma operação de escrita no arquivo retira um setor livre da lista de espaço disponível e efetua a escrita. Em seguida, este novo setor é conectado no fim do arquivo. Para ler um arquivo, simplesmente são lidos os setores seguindo os ponteiros.

Não existe o problema de fragmentação externa com a alocação conectada, pois qualquer setor disponível na lista de espaço livre pode ser usado para atender a um pedido, desde que todos os setores estejam ligados pelos ponteiros. Não existe também a necessidade de declarar o tamanho do arquivo na hora da criação; o arquivo pode ser expandido enquanto existirem setores livres. Portanto, não é necessário compactar o espaço do disco.

A maior desvantagem da alocação conectada é que ela só pode ser usada de modo eficiente para arquivos de acesso sequencial. Por exemplo, para encontrar o i-ésimo setor de um arquivo, é necessário iniciar a procura no início do arquivo e seguir os ponteiros até atingir o i-ésimo setor, sendo que cada acesso a um ponteiro exige uma leitura de disco. Portanto, arquivos com alocação conectada não suportam o acesso direto.

Outra desvantagem da alocação conectada é que o espaço usado para armazenar os ponteiros é descontado do espaço disponível de cada setor, diminuindo o espaço para as informações úteis.

2.6.4 - ALOCAÇÃO INDEXADA

A alocação indexada (ou mapeada) resolve o problema do acesso direto reunindo todos os ponteiros em um único lugar: o bloco de índices. Cada arquivo possui um bloco de índices, o qual é um conjunto de endereços de setores do disco, onde a i-ésima entrada do bloco aponta para o i-ésimo setor do arquivo.

Tais blocos de índices são geralmente chamados "extensões" e podem conter um número qualquer de setores. O descritor contém o endereço do bloco de índices, como pode ser visto na Figura 2.13 (Peterson and Silberschatz, 1983). Inicialmente, todos os ponteiros do bloco de índices não apontam para nenhum setor (são nulos).

A alocação indexada suporta o acesso direto sem os problemas da fragmentação, pois qualquer setor livre do disco pode ser usado para um pedido de alocação de espaço. Entretanto, o uso de setores localizados em pontos arbitrários do disco afeta o tempo de acesso.

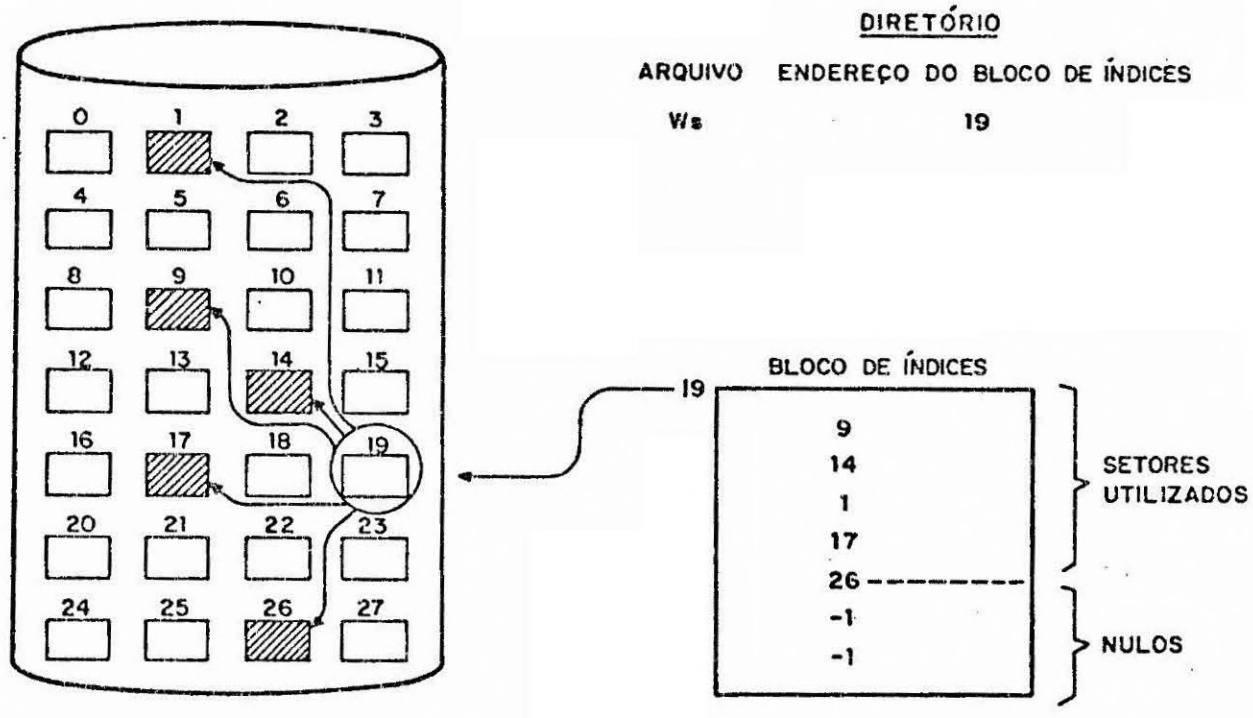


Fig. 2.13 - Alocação indexada.

Outra desvantagem da alocação indexada é a perda de espaço, devido à necessidade de armazenar os blocos de índices. Esta perda de espaço é maior do que no caso da alocação conectada. Se, por exemplo, houver um arquivo que usa apenas dois setores, a alocação conectada perde espaço para armazenar um ponteiro por setor. A alocação indexada perde o espaço de um bloco de índices inteiro, embora somente dois ponteiros não sejam nulos. Surge então o problema de determinar qual o tamanho ideal do bloco de índices.

Como todo arquivo precisa ter um bloco de índices, este deve ser o menor possível. Entretanto, se o bloco de índices for muito pequeno, ele não pode armazenar um número grande de ponteiros, que é necessário para os arquivos maiores. Na prática, um bloco de índices é normalmente um setor físico do disco, o que permite a escrita ou leitura direta pelo "hardware".

Para arquivos de grande porte, vários blocos de índices (extensões) podem ser ligados juntos através de ponteiros. Por exemplo, um bloco de índices pode conter 100 endereços de setores do disco. Caso um arquivo exija maior espaço, o último endereço do bloco de índices pode apontar para outro bloco de índices com mais 100 endereços disponíveis e assim por diante.

Nesse exemplo, utiliza-se a alocação conectada entre as extensões. Uma alternativa é utilizar um bloco de índices separado para apontar para vários blocos de índices, os quais apontam para os arquivos propriamente ditos. Esta alternativa permite aumentar o número de índices para três ou quatro; entretanto, na prática utiliza-se até dois níveis de índices.

Outra possível solução é manter os primeiros "n" ponteiros do bloco de índices (por exemplo: 15) na própria entrada do diretório para o arquivo. Caso o arquivo exija mais do que esses "n" ponteiros, o último deles apontará para um bloco de índices. Deste modo, arquivos pequenos não precisam de um bloco de índices em separado (Peterson and Silberschatz, 1983).

2.6.5 - FRAGMENTAÇÃO E COMPACTAÇÃO

A fragmentação do espaço do disco é um fenômeno causado pela não-utilização de todo espaço livre disponível para o armazenamento de informações. Ela é causada por algumas técnicas de alocação utilizadas, as quais podem causar a fragmentação interna ou externa.

Nos dispositivos de acesso direto, como é o caso dos discos magnéticos, é mais conveniente alocar os arquivos utilizando blocos de tamanho fixo. Este método, entretanto, causa a fragmentação interna, isto é, o último bloco dos arquivos pode ser apenas parcialmente utilizado. Se os blocos forem grandes, a perda de espaço pode ser significativa.

A alocação contínua causa muita fragmentação externa do espaço disponível, enquanto a alocação por lista conectada e a alocação mapeada não apresentam este problema.

A solução mais geral para o problema da fragmentação é a *compactação*, isto é, a reorganização total ou parcial das informações contidas no disco, eliminando temporariamente a fragmentação. Este esquema efetivamente compacta todo o espaço livre do disco em uma única área, mas possui alguns problemas:

- a compactação é lenta e consome muito tempo dos recursos do sistema;

- a compactação normalmente não permite o acesso à área que está sendo compactada, suspendendo o uso do sistema nesse período;

- na compactação é necessário remover todos os ponteiros que se referem aos arquivos que foram compactados, o que nem sempre é possível.

Apesar dos problemas com a compactação, ela é utilizada em vários sistemas, especialmente de maneira parcial, ou seja, é efetuada a intervalos predeterminados como, por exemplo, de um dia ou uma semana (Smith, 1981).

2.6.6 - EFICIÊNCIA E DESEMPENHO

A maior diferença entre os diversos métodos de alocação está na eficiência de armazenamento de cada um.

Existem diferenças também em relação ao tempo de acesso a um dado setor do disco, que podem ser influenciados por algumas características do Sistema Operacional em uso, tais como a maneira de alocar o espaço para o arquivo, o uso de "buffers" na memória principal e a sobrecarga causada pelas computações efetuadas entre os acessos ao disco (Pechura, 1983).

A dificuldade em comparar os vários métodos está em determinar como eles serão usados.

Para qualquer tipo de acesso, a *alocação contínua* requer somente um acesso para ler um setor do disco. Mantendo o endereço inicial do arquivo na memória principal, pode-se calcular facilmente o endereço do i-ésimo setor e acessá-lo diretamente.

Na *alocação conectada*, pode-se também manter o endereço do próximo setor na memória principal e acessá-lo diretamente. Este método é bom para acesso sequencial; entretanto, um acesso direto ao i-ésimo setor poderia exigir "i" leituras do disco.

A *alocação indexada* é mais complexa. Caso o bloco de índices já esteja na memória, o acesso pode ser feito diretamente, porém isto ocupa muito espaço da memória. Se o bloco não estiver disponível, então será necessário ler o primeiro bloco de índices e depois o setor desejado. Para índices de dois níveis são necessárias duas

leituras de blocos de índices. Portanto, o desempenho da alocação indexada depende da estrutura do índice utilizado, do tamanho do arquivo e da posição do setor desejado (Peterson and Silberschatz, 1983).

2.7 - OTIMIZAÇÃO DO DESEMPENHO

Nos sistemas em tempo real, onde é necessário que os dados sejam armazenados e/ou recuperados rapidamente da memória, o uso da memória principal torna-se bastante justificável.

Entretanto, a memória principal utiliza dispositivos semicondutores que possuem um custo/bit bem maior que os discos magnéticos. Isto justifica a escolha do uso de um sistema de discos para o armazenamento dos dados, mesmo num sistema de tempo real.

O disco magnético pode ser usado como um "buffer" de dados de grande capacidade, com um tempo de acesso maior que o da memória principal, porém com um custo/bit bem menor.

O Monitor de Disco deve possuir um alto desempenho em relação ao armazenamento e à recuperação de dados para justificar seu uso num sistema de tempo real.

Justifica-se também a necessidade do aumento do desempenho por duas razões simples:

- tempo de acesso,

- custo das operações de E/S.

Como exemplo, considerando que o tempo necessario para acessar uma posição da memória principal varia, na maioria dos sistemas, entre 50 nanossegundos a 1 microssegundo, enquanto o tempo para ler ou escrever em qualquer tipo de memória secundária (disco, tambor

ou fita magnética) é de no mínimo 10 milissegundos, verifica-se que a razão entre os tempos de acesso varia entre 1.000 e 1.000.000.

Essa enorme relação entre os tempos de acesso implica que, no caso de várias operações de E/S, a Unidade de Processamento - CPU (e praticamente todo o resto do sistema) fica inativa aguardando o término da operações de E/S.

Em segundo lugar, em muitos sistemas, as operações de E/S são caras em termos do tempo de CPU necessário. Os programas de E/S são, em geral, extensos e complexos; envolvem muitas operações relacionadas com a E/S (inicializações de dispositivos, início da E/S, escalonamento dos dispositivos, controle de interrupções, execução da E/S etc.) e necessitam bastante tempo de processamento. Por esta razão, a frequência e o custo das operações de E/S devem ser minimizadas ao máximo (Smith, 1981).

2.7.1 - OTIMIZAÇÃO DO TAMANHO DO BLOCO FÍSICO

O bloco físico é a unidade de transferência de dados utilizada pelo monitor e também a unidade de alocação do disco.

Para a escolha adequada da unidade de alocação do disco, deve-se levar em conta, a priori, a distribuição dos tamanhos dos arquivos do sistema, ou seja, saber a relação entre os vários arquivos e seus tamanhos, bem como as quantidades previstas de arquivos dos vários tamanhos que irão estar ativos no sistema.

Esse parâmetro é altamente dependente tanto de cada aplicação onde o monitor estiver implantado como da carga de trabalho a cada instante, sendo de difícil avaliação (Powell, 1977).

Como orientação geral, existem alguns compromissos na escolha entre os blocos grandes e pequenos. As vantagens dos blocos pequenos são que eles precisam de "buffers" menores e são rapidamente

transferidos. Por outro lado, para processar grandes quantidades de dados, um grande número de pequenos blocos tem de ser acessado e cada acesso é associado a uma sobrecarga do sistema e um tempo de latência extra. Além disto, devido ao uso de marcas ("gaps") entre os vários blocos de um disco, blocos pequenos são ineficientes no uso do espaço físico dos discos.

Essas desvantagens dos blocos pequenos superam as vantagens e, na maioria dos casos (especialmente para processamento de dados na forma sequencial), blocos maiores são normalmente preferidos (Smith, 1981).

2.7.2 - ALOCAÇÃO DOS ARQUIVOS

Além do método de alocação a ser utilizado, outro fator importante para o desempenho é como os arquivos são alocados uns em relação aos outros. Considere-se aqui que um sistema utilize vários discos para o armazenamento dos arquivos.

A alocação dos arquivos pode afetar a eficiência do sistema de duas maneiras:

- alocação em um mesmo disco e

- alocação em discos diferentes.

Se arquivos diferentes estão em uso no mesmo disco, a eficiência pode ser aumentada posicionando os arquivos uns próximos aos outros, o que minimiza as distâncias a serem percorridas pela cabeça do disco ("seek").

Verifica-se que é conveniente a alocação dos arquivos mais usados próximos ao centro do disco e os arquivos menos usados próximos das bordas internas ou externas.

É melhor alocar arquivos que são usados concorrentemente em discos diferentes, pois isto tende a melhorar o tempo de acesso.

Isso pode ser concluído verificando que se um arquivo de acesso sequencial é o único em uso num determinado disco a maioria das operações de E/S irá encontrar a cabeça do disco já posicionada na trilha (ou cilindro) correta, o que evita movimentos desnecessários (Smith, 1981).

2.7.3 - OTIMIZAÇÃO DOS MOVIMENTOS DA CABEÇA DO DISCO

Uma das maiores componentes do tempo de acesso dos discos é o tempo de posicionamento de sua cabeça ("seek time"). Logo, a minimização dos movimentos da cabeça do disco irá melhorar o desempenho.

O método geralmente utilizado para reduzir o tempo de posicionamento é alocar todos os pedidos de E/S para um dado dispositivo numa fila, e selecionar uma ordem de processamento desses pedidos que minimize os movimentos da cabeça do disco. Existem algoritmos típicos como o SSTF ("shortest seek time first"), o SCAN (algoritmo do tipo "elevador") e CSCAN (uma variação aperfeiçoada do SCAN) e outros, os quais serão analisados com maiores detalhes na Seção 2.8. Os objetivos desses algoritmos são (Smith, 1981):

- minimizar o tempo médio de acesso,
- minimizar a variância do tempo de acesso de E/S,
- evitar a discriminação no posicionamento de dados em certas regiões dos discos (exemplo: nas bordas).

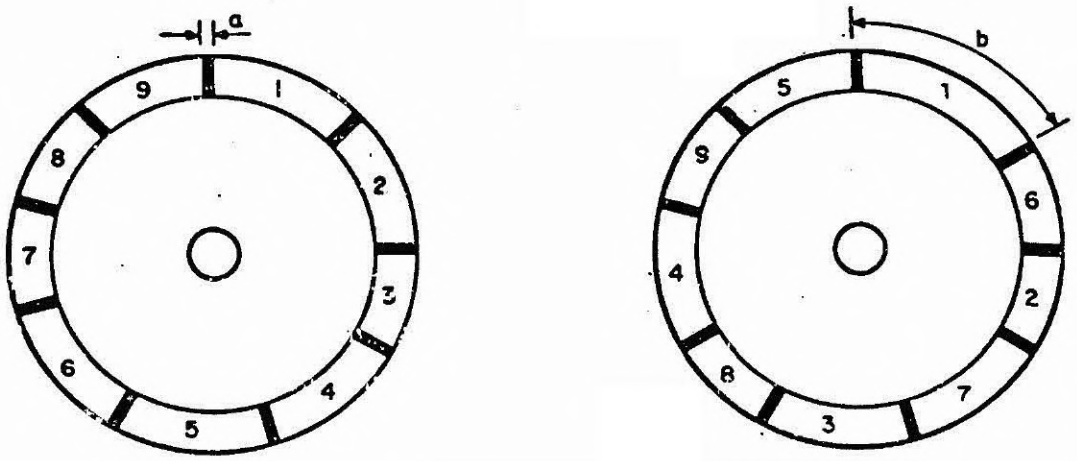
2.7.4 - OTIMIZAÇÃO DOS MOVIMENTOS ROTACIONAIS

Este caso relaciona-se com o escalonamento dos vários pedidos de E/S feitos para uma mesma posição da cabeça, quando se utilizam meios de armazenamento rotacionais, como é o caso dos discos e tambores magnéticos. Para os discos de cabeça móvel, o escalonamento rotacional é normalmente desnecessário, pois é raro ter mais de um pedido de E/S para o mesmo cilindro. No caso dos tambores e discos de cabeça fixa, o escalonamento rotacional é mais importante, sendo chamado "sector queueing".

Os algoritmos para escalonamento já citados também podem ser utilizados nesse caso (por exemplo: SSTF), ou ainda pode existir uma fila de pedidos para cada setor do tambor, atendida na base do "primeiro a chegar-primeiro a ser atendido" ou FCFS ("first come-first served") (Peterson and Silberschatz, 1983).

Outro método muito utilizado para a otimização dos atrasos rotacionais é o entrelaçamento de setores ("sector interleaving"). Ele é utilizado para diminuir a possibilidade de perder um setor durante sua passagem sob a cabeça do disco e ter de esperar uma revolução completa para acessar novamente o setor. Isto ocorre, pois, durante o processamento sequencial de um arquivo, geralmente decorre um tempo entre dois pedidos de E/S, e esse tempo é maior que o tempo para a passagem das marcas ("gaps") entre os setores do disco. Este problema pode ser solucionado com a alocação entrelaçada dos setores do disco, como mostra a Figura 2.14 (Pechura, 1983).

Isso é realizado pulando alguns setores físicos entre cada setor logicamente sequencial. Por exemplo, os 9 setores (1 a 9) seriam numerados na sequência 1, 6, 2, 7, 3, 8, 4, 9, 5. Desse modo, após o sistema ler um setor e, logo em seguida, enviar um pedido para ler o setor seguinte (sequencialmente), é altamente provável que o setor desejado esteja começando a passar sob a cabeça do disco.



a) ESPAÇO ENTRE SETORES LÓGICOS SEM ENTRELAÇAMENTO

b) ESPAÇO ENTRE SETORES LÓGICOS COM ENTRELAÇAMENTO

ENTRELAÇAMENTO DE SETORES

Fig. 2.14 - Entrelaçamento de setores.

Outra técnica para a otimização dos atrasos rotacionais é o chaveamento ou ativação da cabeça do disco quando um processo acessa o último setor de uma trilha e, a seguir, o primeiro bloco da próxima trilha.

O tempo de chaveamento da cabeça é normalmente maior que o tempo de passagem das marcas entre os setores. Se o início for na mesma posição para todas as trilhas, o tempo de latência rotacional será quase sempre o de uma volta completa. Isto é solucionado deslocando o início das várias trilhas consecutivas por 1/4 de trilha, como é o caso dos discos IBM 3310 e 3370. Deste modo o acesso pode continuar na próxima trilha sem perder uma rotação completa (Smith, 1981).

2.7.5 - LEITURA ANTECIPADA E PRÉ-ALOCAÇÃO

O uso de rotinas de leitura antecipada e de pré-alocação é outra alternativa para aumentar o desempenho dos gerenciadores de arquivos, como é o caso do Monitor de Disco.

Como citado em Powell (1977), tais rotinas são utilizadas quando possível, de acordo com o estado do sistema, não devendo influir no funcionamento normal das rotinas básicas.

Duas rotinas utilizadas são a de *leitura antecipada* ("read-ahead") e a de *pré-alocação* ("pre-allocation"). A rotina de leitura antecipada tem a função de trazer antecipadamente para a memória principal alguns blocos adicionais do disco, aproveitando o movimento físico iniciado pelo pedido de leitura de um determinado bloco. Se os arquivos estiverem alocados sequencialmente no disco, é bem provável que a leitura antecipada reduza bastante a necessidade de espera em operações de E/S por parte dos programas de aplicação.

De maneira similar, a rotina de pré-alocação tem a função de alocar antecipadamente alguns blocos, além dos requisitados, caso haja espaço disponível no disco. Para alocação sequencial, tais blocos provavelmente irão ser utilizados e os que não forem necessários serão devolvidos ao sistema. Desta maneira, além de minimizar o número de acessos ao disco, esta técnica diminui a fragmentação dos arquivos num sistema multiprogramado, onde podem ser criados vários arquivos simultaneamente (Powell, 1977).

Ambas as rotinas acima fazem uso intenso de "buffers" da memória principal para agilizar a transferência de dados. Algumas técnicas de transferência de dados serão abordadas a seguir.

2.7.6 - TÉCNICAS DE TRANSFERÊNCIA DE DADOS

Um dos métodos clássicos de aumentar o desempenho do sistema de E/S é a "bufferização" (do inglês: "buffering"). Isto significa fazer a transferência de dados da/para a memória secundária através de áreas da memória principal que não são usadas pelo programa: os "buffers". As influências no desempenho são (Powell, 1977):

a) Reduz o tempo necessário para execução de um programa, permitindo que ele possa ser executado em instantes que, sem ela o programa seria forçado a esperar pela E/S. Esta característica é muito importante em sistemas aplicados ao controle de processos em tempo real.

b) Permite a liberação mais rápida de todos os recursos alocados ao programa, devido à diminuição de seu tempo de execução.

c) Aumenta individualmente o desempenho de muitos tipos de dispositivos de E/S. Por exemplo, sob certas circunstâncias, seria necessário 30ms para que um acionador de discos estivesse pronto para transferir um bloco de dados e 3ms para a transferência propriamente dita. Se um segundo bloco tiver de ser transferido logo após o primeiro, o tempo para que o acionador fique pronto para o segundo bloco não será mais necessário, o que reduz significativamente o tempo de espera total (Freeman and Perry, 1977).

A "bufferização" tem também suas limitações, que são as seguintes:

- se um programa fornecer ou requerer dados mais rapidamente do que o dispositivo de E/S por um longo período de tempo, as capacidades dos "buffers" serão excedidas e o ganho no desempenho tornar-se-á nulo.

- Para determinados requisitos de E/S a "bufferização" não é possível. Por exemplo, quando um programa requisita registros de um arquivo numa sequência imprevisível, não existe maneira do sistema de E/S antecipar um pedido. Neste caso, a manipulação de dados extra exigida pela "bufferização" pode ser prejudicial ao desempenho.

A Figura 2.15 mostra as vantagens e as limitações da "bufferização" em alguns casos típicos (Freeman and Perry, 1977). Tais casos estão aí ilustrados de maneira simplificada, ignorando os efeitos de outros mecanismos de controle do desempenho sobre os dispositivos de E/S.

Para os casos simples da Figura 2.15, pode-se concluir que o uso de dois "buffers" possibilita o máximo ganho, pois, com um único "buffer" ocorre a alternância entre E/S e o processamento.

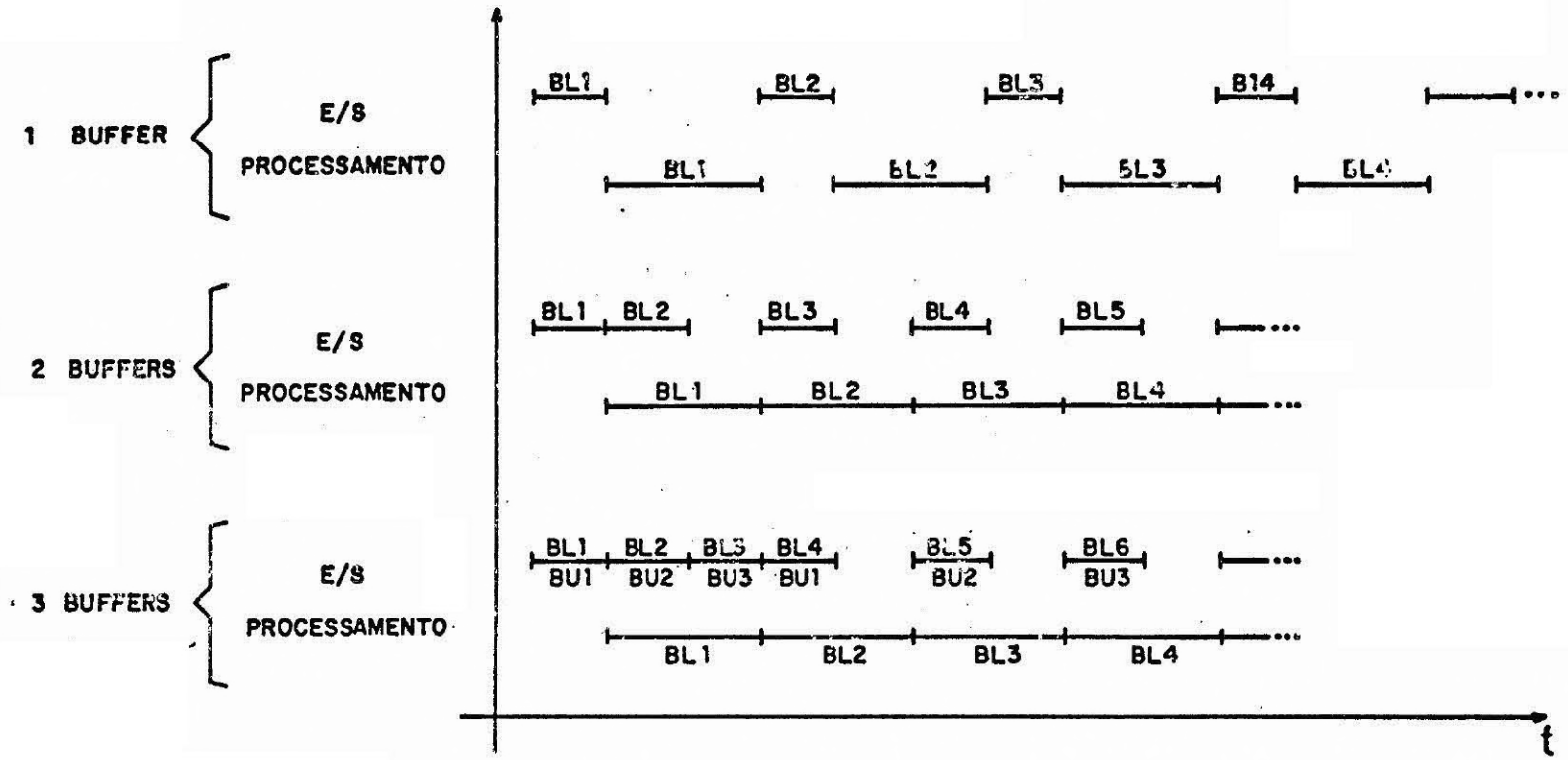
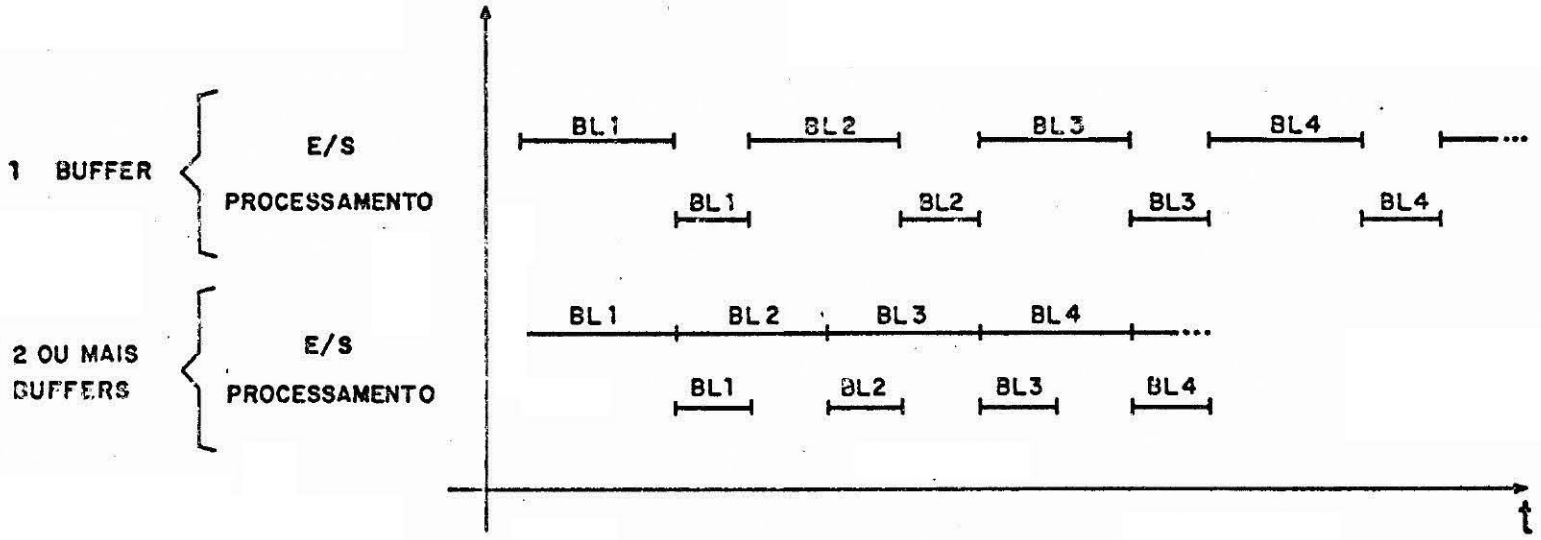


Fig. 2.15 - "Bufferização" simples.

Fig. 2.15 - Conclusão



CASO 2 : TEMPO DE E/S > TEMPO DE PROCESSAMENTO

Com dois "buffers" ocorre processamento contínuo (caso 1) ou E/S contínua (caso 2). Verifica-se também que o uso de três ou mais "buffers" não fornece nenhuma vantagem adicional em relação ao uso de dois "buffers".

Podem existir casos em que o uso de três ou mais "buffers" é justificado, como mostrado nos exemplos a seguir:

a) *Tempo de processamento variável:* se o tempo de processamento varia de bloco para bloco, um grande número de "buffers" pode ter um efeito nivelador, fornecendo blocos de entrada ou aceitando blocos de saída rapidamente, de uma só vez. Um programa tem um tempo de processamento variável, caso ele processe somente certas classes de registros de um arquivo sequencial.

b) *Tempo de E/S variável:* como no exemplo acima, um grande número de "buffers" também pode conseguir um efeito nivelador neste caso. O tempo de E/S pode variar de bloco para bloco devido à sua recuperação aleatória ou devido a mecanismos de controle dos dispositivos, efetuados continuamente (por exemplo: algoritmo de escalonamento).

c) *Efeitos do ambiente:* em ambientes complexos (por exemplo, multiprogramação) é vantagoso, sempre que possível, executar várias operações de uma só vez. Por exemplo, o preenchimento de vários "buffers" de entrada forma uma fonte de dados que é vantajosa quando, repentinamente, o canal de entrada é tomado por uma tarefa concorrente. O preenchimento de vários "buffers" de saída permite a operação contínua de um dispositivo de saída, enquanto a CPU é tomada por uma tarefa de maior prioridade.

Powell (1977) mostra um estudo do número de "buffers" adequado para conseguir um determinado desempenho, no caso do computador CRAY-1. Nesse caso, conclui-se que o número de "buffers" deve ser menor que oito para obter um bom desempenho.

Algumas das formas de "bufferização" simples são analisadas por Freeman e Perry (1977) e descritas a seguir:

1) *Simples:* utiliza dois "buffers" de maneira alternada. Enquanto um é preenchido, o outro é processado. Pode ser usada com um número maior de "buffers".

2) *Troca de funções ("exchange buffering"):* os "buffers" e as áreas do programa são intercambiados em suas funções, sem haver a transferência de dados entre tais regiões.

3) *Dinâmica:* os "buffers" são atribuídos aos processos em tempo de execução, conforme a necessidade.

4) *Circular:* utiliza um segmento contínuo da memória de tamanho igual ou maior que o maior bloco de dados a ser processado. Este segmento é tratado como se fosse infinito, isto é, o final do segmento é como o início.

2.7.7 - TÉCNICAS DE MEMÓRIA "CACHE"

Uma técnica mais elaborada de utilização de "buffers" como interface na transferência de dados entre os periféricos e os processos é conhecida como memória "cache".

A palavra "cache" é derivada do francês, que significa memória "escondida" ou "transparente", no sentido de que para os processos é como se os dados fossem transferidos dos/para os periféricos diretamente.

Nesta técnica, são mantidas cópias de algumas partes do disco nos "buffers" destinados ao "cache". Sempre que um pedido de acesso ao disco se referenciar aos dados localizados no "cache", a transferência poderá ser feita rapidamente entre o "buffer" e a área

de dados do processo, sem necessidade de acessar o disco. Caso contrário, precisa-se esperar o tempo de acesso necessário exigido pelo disco.

Quando a parte do disco desejada é encontrada na memória "cache", ocorre um sucesso ("hit"), caso contrário ocorre um insucesso ("miss").

Se a razão entre sucessos e insucessos ("hit/miss ratio") é alta, então o tempo médio de acesso ao periférico é reduzido, aumentando o desempenho. Todos os métodos de memória "cache" baseiam-se neste conceito.

O uso de uma técnica de memória "cache" envolve a solução de dois problemas principais: o primeiro é como determinar rapidamente se o "cache" contém a informação desejada por um processo. Este problema é mais sério quando se utiliza essa técnica entre a memória principal e uma memória rápida (usada como "cache"), onde os tempos de decisão devem ser bem pequenos. No caso de "cache" entre a memória e um periférico, este problema não é de grande importância.

O segundo problema é mais crítico: qual é o melhor algoritmo a ser usado para maximizar as chances de que a informação pedida por um processo esteja disponível no "cache" quando desejado? Na prática é impossível prever com exatidão todos os pedidos que serão feitos pelos processos.

2.7.7.1 - EFEITOS DO "CACHE" NAS OPERAÇÕES DE LEITURA

Existem basicamente duas classes de algoritmos que tentam fornecer o melhor meio de escolher quais dados devem ser mantidos no "cache" e quando removê-los para serem trazidos outros dados do disco:

1) *Algoritmos de leitura antecipada ("prefetch"):* Consideram que os dados sejam armazenados de modo sequencial e que um acesso a uma determinada parte dos dados implica, com uma grande probabilidade, que a parte sequencialmente seguinte a anterior será logo acessada.

2) *Algoritmos de substituição ("replacement"):* Baseiam-se no fato de que quaisquer dados a serem trazidos para a memória devem substituir outros dados que já estejam lá. Deve-se então deslocar os dados que provavelmente não serão utilizados num futuro próximo, com base em algumas suposições sobre o uso anterior e futuro de um determinado conjunto de dados.

Na prática, os algoritmos de leitura antecipada tentam determinar quais dados devem ser trazidos para a memória "cache" antes que os mesmos sejam pedidos, e os algoritmos de substituição tentam determinar quais dados devem permanecer no "cache", uma vez que já foram colocados lá.

É claro que os dois tipos de algoritmos interagem entre si, pois, sempre que se busca novos dados, deve haver o deslocamento dos dados anteriormente localizados no "cache".

Outra importante decisão no projeto de um sistema que utiliza uma memória "cahe" é a determinação da quantidade de memória a ser utilizada pelo "cache" em relação ao tamanho da memória principal.

Deve-se também utilizar "buffers" com um tamanho bem maior que a unidade básica de informação (1 "byte" ou palavra), de modo que a sobrecarga do processamento das decisões do algoritmo seja amortizada nas transferências de grandes quantidades de dados e as decisões não precisem ser tomadas frequentemente. Normalmente a unidade de acesso do "cache" é igual a vários setores do disco, isto é, um bloco físico ("cluster").

Tomando como base a estrutura do monitor adotada neste trabalho, o "cache", pode ser implementado em dois níveis, como mostra a Figura 2.16.

a) *Nível 3 - S.O.A:* neste caso o S.O.A tem informações sobre quais registros lógicos pertencem a um determinado arquivo e também quais setores físicos correspondem aos registros lógicos. Deste modo, é mais fácil prever quais registros devem ser requisitados e quais devem ser substituídos ou mantidos no "cache". Por outro lado, a implementação é mais complexa devido ao maior número de interações com outros níveis da estrutura.

b) *Nível 2 - C.E.S:* neste caso o C.E.S. não tem conhecimento da estrutura dos arquivos. A decisão sobre quais registros devem ser lidos ou substituídos é feita mecanicamente. Podem ocorrer erros e serem acessados registros que não pertençam ao arquivo em questão. Este fator leva a algoritmos menos eficientes, porém mais simples e independentes dos demais níveis da hierarquia. Outra vantagem desta alternativa é que o C.E.S. pode ser mais facilmente modificado no caso de haver necessidade de modificação do algoritmo ou de substituição do "hardware" que implique alteração do "software". Este último fator contribui para aumentar a portabilidade do monitor.

O aumento de desempenho obtido com o uso de técnicas de memória "cache" pode ser verificado observando um exemplo prático, como o descrito por Chaney e Johnson (1984). Nesse caso, mostra-se a influência da relação sucessos/insucessos ("hit/miss ratio") sobre a taxa de transferência de dados entre a memória e o disco.

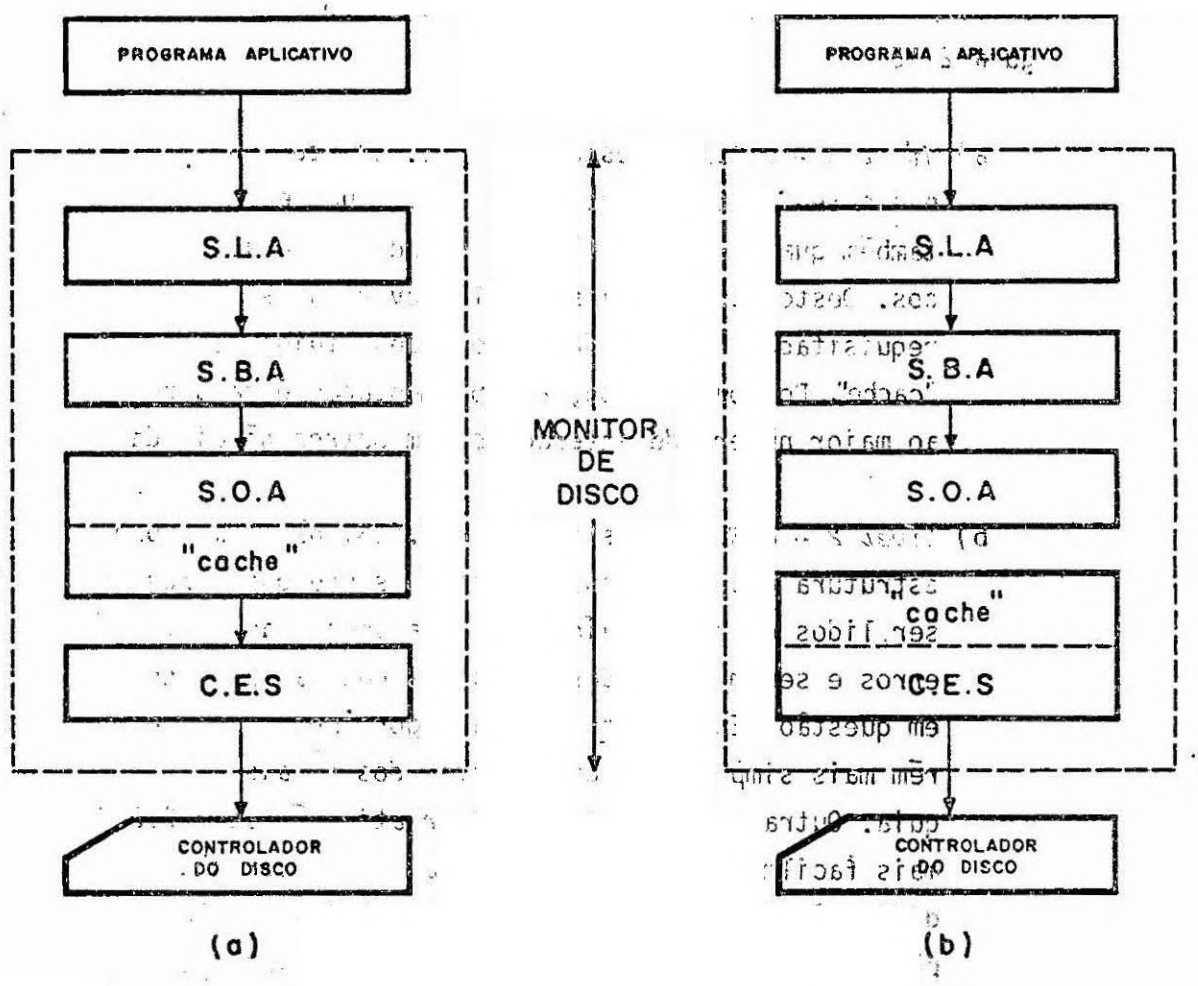


Fig. 2.16 - Modos de implementação do "cache".

O exemplo é baseado no Sistema Operacional PC-DOS 2.0, onde utiliza-se uma memória "cache" de 256 "kbytes" e considera-se um bloco físico de 1024 "bytes" e uma taxa de transferência de dados entre a memória principal e o "cache" de 350 "kbytes"/segundo. Os resultados obtidos são mostrados na Figura 2.17 (Chaney and Johnson, 1984), a qual mostra a taxa de transferência efetiva do disco para a memória versus a relação sucessos/insucessos.

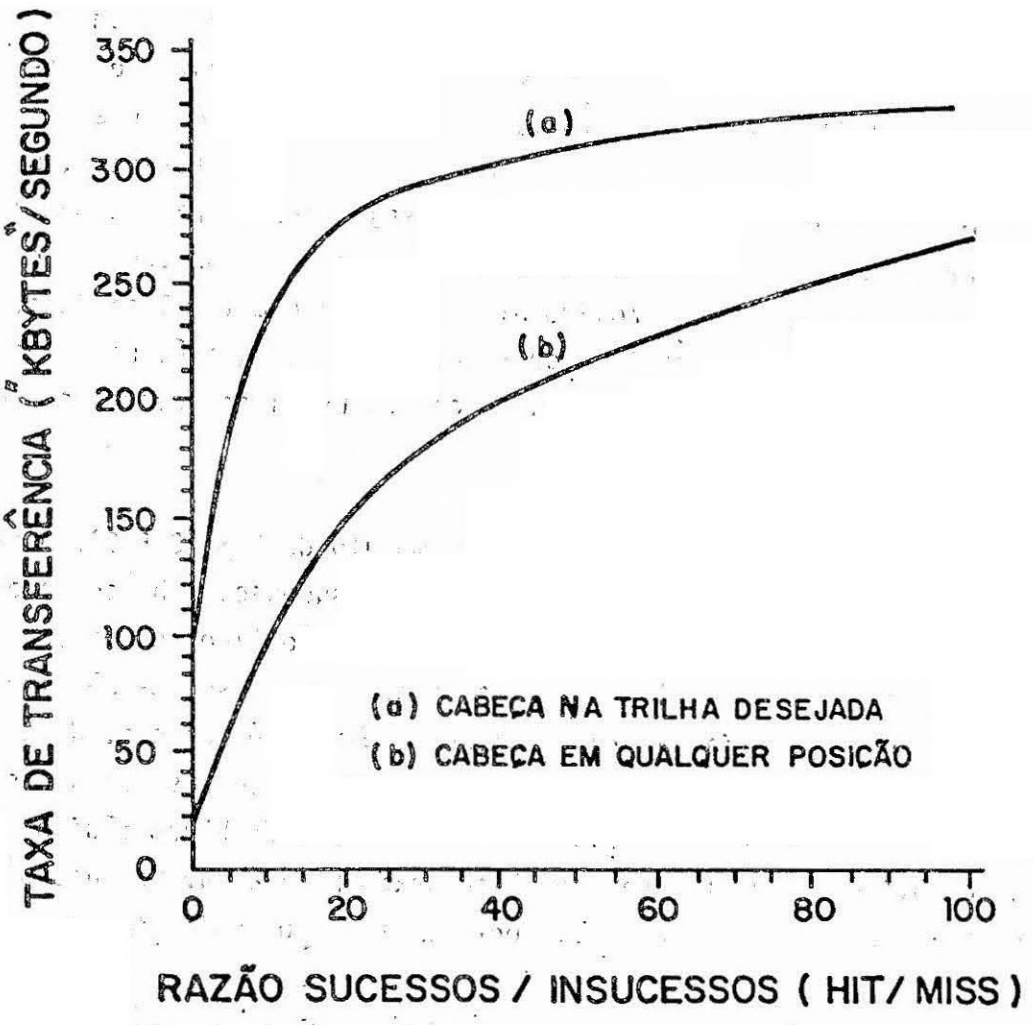


Fig. 2.17 - Efeitos do uso da memória "cache".

Conclui-se que quanto maior essa relação, maior é a taxa de transferência obtida. A relação acima é função direta do tamanho da memória destinada ao "cache", porém quanto maior o "cache", maior o custo do sistema.

A curva (a) da Figura 2.17 ilustra o caso em que a cabeça do acionador de discos já se encontra na trilha adequada e o único atraso é o rotacional. A curva (b) mostra o tempo médio de acesso, incluindo o tempo médio necessário para a busca da trilha desejada.

Os *algoritmos de leitura antecipada* variam em função do modo como o "cache" é implementado. Se o "cache" é parte do sistema de organização de arquivos (S.O.A.), o algoritmo normalmente é projetado para que sempre que alguma parte de um arquivo seja acessada, o próximo bloco físico ("cluster") do arquivo seja trazido para a memoria, caso ainda não esteja lá, não importando se ocorreu um sucesso ou insucesso. Note-se que este próximo bloco pode não estar localizado fisicamente adjacente ao que foi acessado. Deste modo, após um insucesso ao acessar o arquivo pela primeira vez, os próximos acessos serão feitos com sucesso.

Se o "cache" é parte do módulo de E/S (C.E.S), o algoritmo é mais simples: sempre que ocorrer um insucesso, o bloco desejado e mais alguns blocos fisicamente adjacentes serão lidos do disco, considerando-se que blocos fisicamente contíguos devem ser também logicamente contíguos, ou seja, a fragmentação do espaço é pequena. Neste caso, pode ocorrer que, ocasionalmente, fragmentos de outros arquivos sejam trazidos do disco para o "cache", se o tamanho do bloco físico acessado for diferente do tamanho do bloco usado para armazenamento do arquivo. Este método torna-se ineficiente para discos muito fragmentados.

Os *algoritmos de substituição* utilizam três técnicas básicas:

- substituição aleatória,
- primeiro a entrar-primeiro a sair (FIFO),
- menos usado recentemente (L.R.U.).

No primeiro caso, os dados a serem deslocados do "cache" são escolhidos aleatoriamente, sem qualquer preocupação com a probabilidade de uso dos dados.

No método FIFO, o algoritmo sempre desloca os dados que estão mais tempo no "cache".

O método L.R.U é uma variação do FIFO; entretanto, os dados não utilizados a mais tempo é que são deslocados. Este método baseia-se na suposição de que os dados usados frequentemente pelo processo devem permanecer no "cache", substituindo-se os menos usados.

De acordo com Chaney e Johnson (1984), o método L.R.U é o mais eficiente para o uso da memória "cache" com discos magnéticos. Isto se deve ao fato de que certas áreas do disco como o diretório e as tabelas de alocação são usadas extensivamente pelos processos, sendo importante que estas áreas estejam no "cache" todo o tempo.

2.7.7.2 - EFEITOS DO "CACHE" NAS OPERAÇÕES DE ESCRITA

Considere-se agora como o método "cache" afeta as operações de escrita no disco. Deve o "cache" ser envolvido nas operações de escrita? Sim, o "cache" deve estar envolvido, pois ele representa os dados presentes no disco.

Um dos principais requisitos de qualquer método de memória "cache" é que ele seja uma cópia exata da parte do disco que ele representa. Se uma parte do disco for modificada por uma operação de escrita, deve-se verificar se aquela parte do disco também estará no "cache". Se estiver, existem duas opções: uma delas é considerar a parte recém-escrita como não estando mais no "cache", o que evita que o processo pegue dados errôneos do "cache".

A segunda opção é atualizar a cópia da parte escrita que fica no "cache", de modo a refletir as modificações do disco. Na verdade é mais conveniente transferir os dados do processador para o "cache" e daí para o disco.

Existem basicamente dois algoritmos usados para a escrita no disco. No método de *escrita direta* ("write-through"), o processador transfere a informação a ser escrita para o "cache" e, em seguida, pede para que tal informação seja escrita no disco. Logo após fazer o pedido de escrita, o processador pode continuar com outras tarefas, aguardando uma interrupção do periférico, a qual indica o fim da escrita. Este método tem uma resposta muito boa; entretanto, existe um curto período de tempo em que o processo aplicativo pensa que os dados foram escritos com sucesso no disco, quando na realidade isto ainda não ocorreu.

Se o computador for desligado antes da transferência ocorrer, haverá a perda de informação. Outro problema é que quaisquer erros de escrita que ocorram devem ser tratados pelo Sistema Operacional ou pelo programa de E/S, pois o processo pensa que a escrita foi executada com sucesso.

Porém no caso do uso de disco Winchester, a confiabilidade é alta e as taxas de erro bem baixas. O ganho de desempenho obtido justifica o tratamento dos erros que ocorrerem na escrita.

O outro método é o da *escrita defasada* ("write-back"). Neste caso, os dados do "cache" não são copiados para o disco até que o algoritmo de substituição determine que tais dados devem ser deslocados para dar lugar a outros dados. Normalmente, cada "buffer" do "cache" possui um indicador de tipo. Se o "buffer" for do tipo "escrita", isto significa que ele foi atualizado, mas não copiado no disco. Estes "buffers" de escrita ficam no "cache" até que precisem ser substituídos por outros.

Geralmente, após um determinado intervalo de tempo, o sistema pode copiar todos os "buffers" de escrita pendentes no disco. Isto minimiza a perda de dados no caso de um problema de "hardware" acidental, deixando o sistema menos vulnerável.

O tempo médio para escritas no disco pode ser reduzido observando um aspecto importante: é mais eficiente escrever vários "buffers" de uma só vez do que "buffers" muito espaçados entre si. Como as escritas no disco são dependentes dos pedidos de cada processo, a única opção para aumentar o desempenho é escolher uma sequência ótima para a escrita dos "buffers" no disco.

Na hora da escrita no disco, pode-se então utilizar um algoritmo de escalonamento dos pedidos que minimize o movimento médio do braço do acionador de discos. Este algoritmo de escalonamento vai então trabalhar em conjunto com o método "cache", como é mostrado em Mckeon (1985).

2.8 - ESCALONAMENTO DO DISCO

O tempo de acesso de um disco é composto de três partes: do tempo de busca da trilha ("seek time"), do tempo de espera pelo bloco desejado, ou tempo de atraso rotacional ("latency time"), e do tempo de transferência dos dados do disco para a memória ("transfer time"). O Sistema Operacional pode aumentar o desempenho do dispositivo acionador através da ordenação adequada dos vários pedidos de acesso ao disco. Esta ordenação é conhecida como escalonamento dos pedidos ("scheduling") e visa a minimização do tempo de busca e do tempo de espera pelo bloco.

Uma causa comum da ineficiência na implementação de sistemas multiprogramados pode ser atribuída ao uso ineficiente do sistema de discos. Para dispositivos de cabeça fixa (discos e tambores) existem técnicas de "hardware" e "software" que otimizam a eficiência e o tempo de resposta simultaneamente. Para os dispositivos de cabeça móvel, isto é, com movimentos do braço, não existe uma técnica que otimize esses dois parâmetros ao mesmo tempo, ou ainda, otimize um dos parâmetros por toda a faixa de condições de operação. Nesses casos, pode ser necessário o uso de duas técnicas de otimização que operem de forma conjunta, como sugere Teorey e Pinkerton (1972).

2.8.1 - ESCALONAMENTO DO TIPO FCFS

A forma mais simples de escalonamento é a do tipo "primeiro a chegar-primeiro a ser atendido", em inglês, FCFS ("first-come-first-served"). Este algoritmo é simples, intrinsicamente justo e possui algumas desvantagens. O FCFS possui um comportamento de busca de trilhas aleatório, isto é, os pedidos são atendidos na ordem de chegada na fila de pedidos. Além disto, ele não leva em conta as relações posicionais entre os pedidos de E/S da fila, nem a posição atual da cabeça de leitura/escrita (Teorey and Pinkerton, 1972).

Como exemplo considere-se uma fila de pedidos de acesso para as seguintes trilhas: 98, 183, 37, 122, 14, 124, 65 e 67, onde o primeiro pedido é para a trilha 98 e o último para a trilha 67. Supondo que a cabeça esteja posicionada inicialmente na trilha 53, essa irá se mover da trilha 53 para a 98, depois para a 183, 37, 122, 14, 124, 65 e 67 com um movimento total de 640 trilhas, como mostrado na Figura 2.18 (Peterson and Silberschatz, 1983).

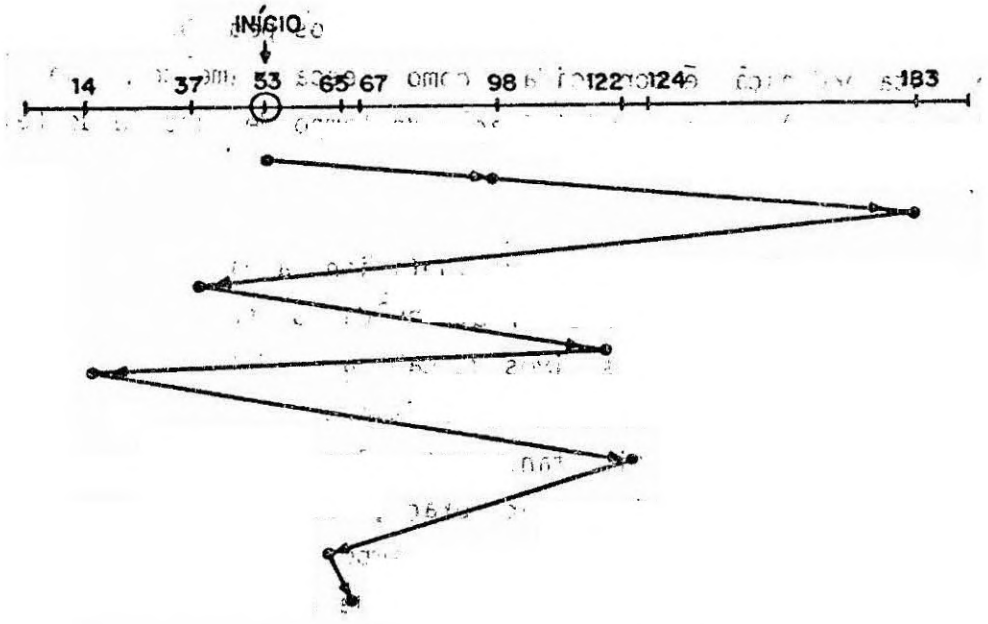


Fig. 2.18 - Escalonamento FCFS.

Pelo exemplo, verifica-se uma enorme perda de tempo com movimentos desnecessários da cabeça, como é o caso quando a cabeça está na trilha 122, move-se para a trilha 14 e logo em seguida volta para a trilha 124. Se os pedidos de acesso para as trilhas 37 e 14 pudessem ser atendidos juntos, antes ou depois dos pedidos para as trilhas 122 e 124, o movimento total da cabeça seria bastante reduzido, o que melhoraria a resposta do disco.

2.8.2 - ESCALONAMENTO DO TIPO SSTF

O escalonamento do tipo SSTF ("shortest-seek-time-first") atende a todos os pedidos para as trilhas próximas à posição atual da cabeça, tentando com isto minimizar os movimentos desta. O algoritmo seleciona o pedido com o menor tempo de busca a partir da posição atual da cabeça.

Para o mesmo exemplo anterior (Seção 2.8.1) o pedido mais próximo da posição inicial da cabeça (trilha 53) é para a trilha 65; em seguida para as trilhas 67, 37, 14, 98, 122, e 183, como mostra a Figura 2.19. Este método de escalonamento resulta em um movimento total da cabeça de apenas 236 trilhas, ou seja, um pouco mais de um terço da distância necessária para o algoritmo FCFS (Peterson and Silberschatz, 1983).

O método SSTF utiliza as relações posicionais entre a cabeça do disco e os pedidos de E/S para melhorar a resposta; entretanto, ele faz uma discriminação entre pedidos individuais. Suponha o caso em que existe uma fila de pedidos de tamanho constante "L". Uma vez que o braço do disco esteja se movimentando em uma dada direção, a densidade dos pedidos imediatamente na frente dele é baixa. Mesmo que o número total de pedidos torne-se maior atrás do braço, o algoritmo SSTF só é dependente do pedido mais próximo da posição atual do braço. Consequentemente, o braço não será necessariamente movido para a direção onde existem mais pedidos. Ocasionalmente, pode ocorrer um novo pedido que muda a direção do movimento do braço, embora a densidade de pedidos na região indique a necessidade de manter a direção original (Teorey and Pinkerton, 1972).

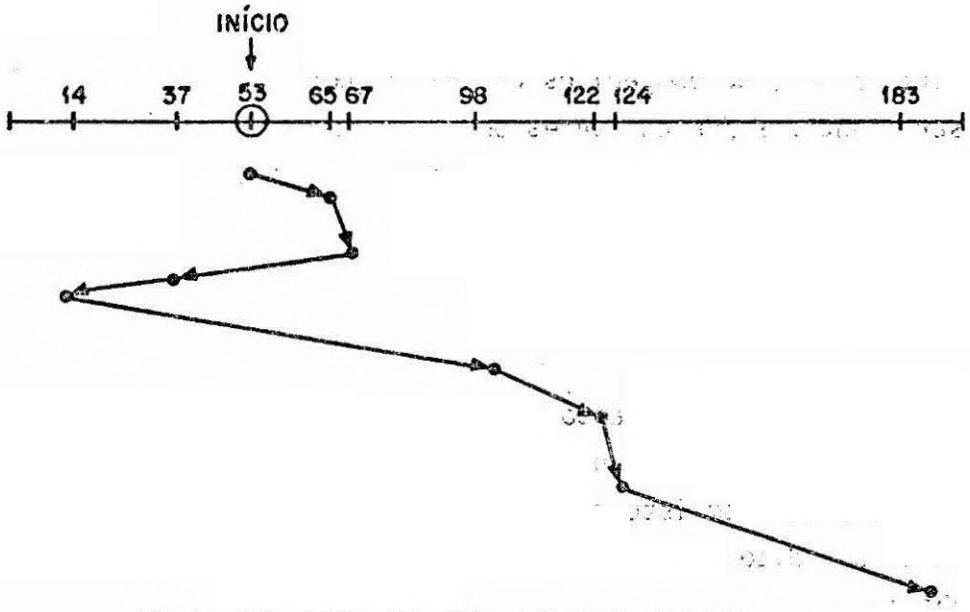


Fig. 2.19 - Escalonamento SSTF.

No caso do exemplo anterior, considerando que dois pedidos na fila para as trilhas 14 e 186, se chegar um novo pedido para uma trilha próxima da trilha 14, enquanto esta estiver sendo atendida, este novo pedido será atendido, fazendo com que o pedido para a trilha 186 fique aguardando. Pode ocorrer que outros pedidos próximos ao que está sendo atendido cheguem na fila, fazendo com que, na teoria, o pedido para a trilha 186 fique esperando indefinidamente (Peterson and Silberschatz, 1983).

2.8.3 - ESCALONAMENTO DO TIPO SCAN

O método SCAN, proposto por Denning (1967), é uma alternativa para o método SSTF e não faz a discriminação entre alguns pedidos individuais como ocorre com o SSTF. O SCAN pode ser visto como o método básico para aumentar a eficiência do escalonador do disco, pois leva em conta a natureza dinâmica da fila de pedidos.

Neste método, a cabeça de leitura e escrita inicia o movimento em uma das extremidades do disco e move-se em direção à outra extremidade, atendendo aos pedidos conforme vai atingindo cada trilha. A direção do movimento da cabeça é revertida quando não houver mais pedidos na direção atual do movimento. O SCAN é também chamado algoritmo "elevador" (ou ainda "LOOK"), pois tem o comportamento semelhante ao de um elevador que atende aos andares de um edifício (Peterson and Silberschatz, 1983).

O maior problema com este método é que ele causa uma variância muito alta nos tempos de espera de alguns pedidos individuais, pois não faz discriminação entre novos pedidos e pedidos já pendentes na fila. Se um pedido de uma trilha logo adiante da posição atual da cabeça chega na fila, ele será atendido quase que imediatamente, enquanto um pedido para uma trilha que esteja logo atrás da cabeça tem que esperar que esta se mova até o final do disco e retorne antes deste pedido ser atendido.

Considerando uma distribuição uniforme de pedidos para todas as trilhas, verifica-se que a densidade dos pedidos imediatamente atrás do movimento da cabeça é baixa, pois estas trilhas foram atendidas recentemente. A maior densidade de pedidos está na extremidade do disco oposta ao movimento da cabeça; estes pedidos também irão esperar um tempo maior para ser atendidos. A Figura 2.20 mostra a atuação do algoritmo SCAN no exemplo anterior, supondo que a cabeça inicie o movimento na trilha 53 indo em direção à trilha 0. Neste caso, a cabeça irá para a trilha 37 e 14, mudará de direção e irá atender as trilhas 65, 67, 98, 122, 124 e 183. (Peterson and Silberschatz, 1983).

Uma variação do escalonamento SCAN é o escalonamento "N-STEP SCAN", cuja finalidade é diminuir a variância dos tempos de espera individuais, não degradando o desempenho significativamente. Neste caso, o braço do disco movimenta-se como o SCAN, porém todos os pedidos que chegam durante o movimento em uma direção são agrupados e reordenados para um atendimento ótimo durante o movimento de retorno

do braço. Os pedidos são agrupados em conjuntos de tamanho "N" ou menores, ou seja, o mínimo entre "N" e o atual comprimento da fila de pedidos, logo após o atendimento do grupo anterior. A variância menor é resultante do fato de que, uma vez formado um grupo de pedidos, não é permitido um novo reordenamento deles (Teorey and Pinkerton, 1972).

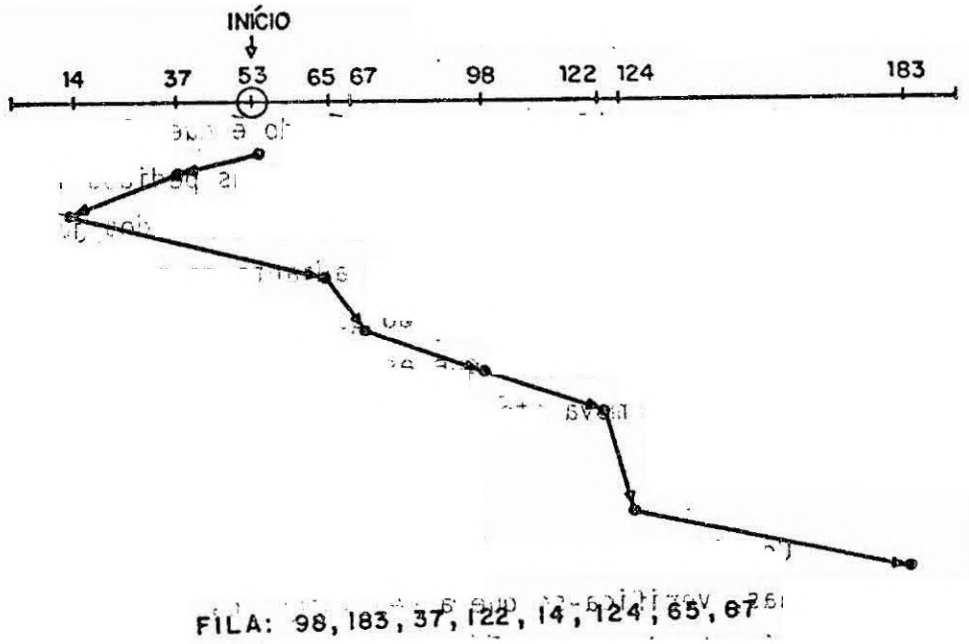


Fig. 2.20 - Escalonamento SCAN (ou "LOOK")

Outra variação do SCAN é o escalonamento C-SCAN ("Circular-SCAN"), que é uma tentativa de diminuir a variância do tempo de espera sem degradar a eficiência ou aumentar o tempo de resposta (Teorey, 1972). O algoritmo C-SCAN move a cabeça do disco de uma extremidade a outra, atendendo os pedidos enquanto se movimenta numa dada direção. Quando não existirem mais pedidos para posições adiante da cabeça, o braço retorna imediatamente ao ponto inicial, não atendendo nenhum pedido no caminho de volta. O C-SCAN trata o disco como se ele fosse circular, com a última trilha adjacente à primeira. A Figura 2.21 mostra o exemplo anterior para o caso do algoritmo C-SCAN, também chamado "C-LOOK" (Peterson and Silberschatz, 1983).

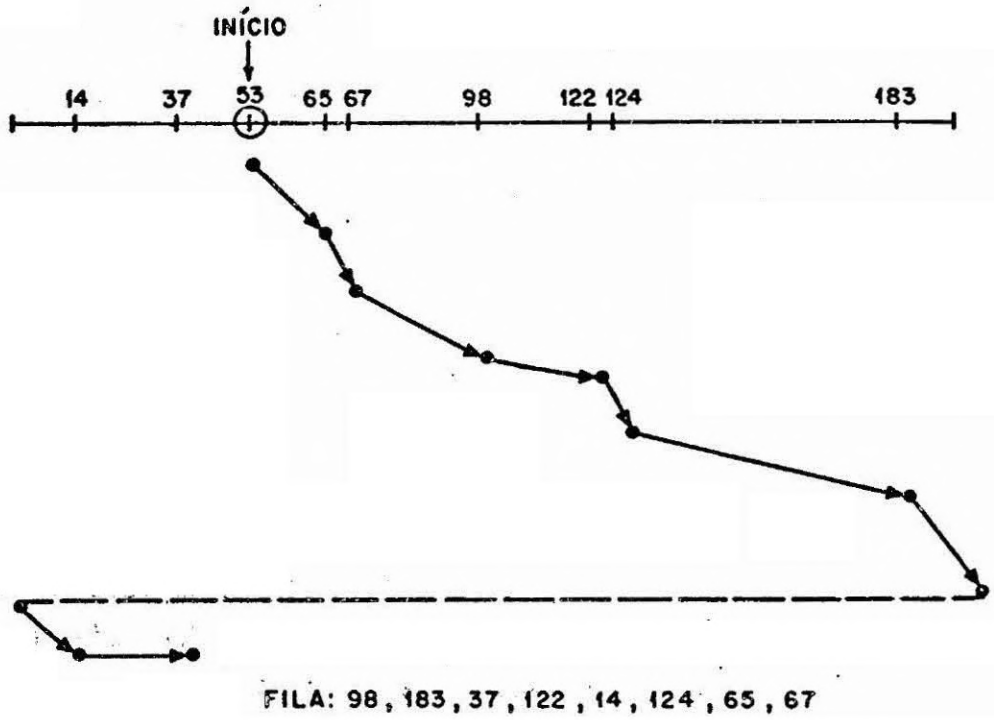


Fig. 2.21 - Escalonamento C-SCAN (ou C-LOOK).

2.8.4 - ESCALONAMENTO DO TIPO ESCHENBACH

Esse método foi projetado inicialmente para sistemas de transmissão de mensagens, os quais operam normalmente sob pesadas condições de carga. O movimento do braço efetua uma varredura circular como a do caso do C-SCAN, porém existem várias diferenças entre os dois. Define-se uma varredura do tipo Eschenbach de ordem "E" quando esta possui "E" voltas do disco por cilindro, com m/E setores atendidos por volta (m = número total de setores por trilha). Numa varredura de ordem E, E-1 setores são pulados em cada setor atendido, o que significa que:

a) De modo a atender todos os m setores de um cilindro, o braço deve permanecer na mesma posição em, no mínimo "E" voltas do disco.

b) Para os cilindros 0, 1,..., C-1, o braço pode gastar até "E" voltas no cilindro 0 antes de mover-se para o cilindro 1, "E"

voltas no cilindro 1, e assim por diante até gastar "E" voltas no cilindro N. A partir daí o braço move-se diretamente para o cilindro 0, sem parar em nenhum cilindro intermediário.

A partir de (a) e (b), conclui-se que o escalonamento Eschenbach possui uma característica de otimização rotacional intrínseca, no sentido de que todos os setores possíveis são atendidos em uma mesma volta e o atendimento é feito na ordem em que os setores são encontrados (Teorey and Pinkerton, 1972).

2.8.5 - OUTRAS CONSIDERAÇÕES SOBRE ESCALONAMENTO

Uma causa comum da ineficiência na implementação de sistemas multiprogramados pode ser atribuída ao uso ineficiente do sistema de discos. Para dispositivos de cabeça fixa (discos e tambores) existem técnicas de "hardware" e "software" que otimizam a eficiência e o tempo de resposta simultaneamente.

Para dispositivos de cabeça móvel, não existe uma técnica que otimize os dois parâmetros, citados acima, ao mesmo tempo, ou ainda otimize um deles por toda a faixa das condições de operação.

Nesses casos, é necessário o uso de duas técnicas de otimização que operem de forma conjunta (Teorey and Pinkerton, 1972).

Em vários artigos disponíveis na literatura (Frank, 1969; Teorey and Pinkerton, 1972: Gotlieb et alii, 1973; Denning, 1967), são propostos e analisados matematicamente vários métodos de escalonamento disponíveis para a otimização do funcionamento dos acionadores de disco. Entretanto, existem várias outras condições impostas na prática que diminuem o efeito de tais métodos em relação a um escalonamento simples, do tipo FCFS. Deve ser levada em conta a influência causada pelo Sistema Operacional, pelo "software" e pelo "hardware" em uso sobre o método de escalonamento (Teorey, 1972).

Em geral, o escalonamento é mais benéfico quando é implementado como um dos componentes do projeto do Monitor de Disco e não como um algoritmo independente. Os principais fatores que afetam a eficiência do escalonamento, de acordo com Teorey (1972), são:

a) os dispositivos de armazenamento em disco são um recurso limitante no sistema;

b) o nível de multiprogramação do sistema;

c) o uso de múltiplos subsistemas de disco;

d) a distribuição de pedidos não-uniforme;

e) os métodos de organização dos arquivos;

f) o tempo de procura ("seek time") que não é fator dominante no tempo de acesso total.

A seguir, faz-se uma breve descrição de cada um dos fatores citados acima.

a) *Disco como recurso limitante*

Em alguns sistemas multiprogramados, o congestionamento do sistema pode não estar sendo causado pelos dispositivos de armazenamento em disco. Neste caso, o escalonamento não deve ser considerado importante, mas devem ser concentrados os esforços na otimização ou substituição do componente do sistema que está causando um baixo desempenho. Quando se confirma que o sistema de discos está causando problemas, o escalonamento pode ser usado como um meio de aumentar a resposta ("throughput").

b) *Nível de multiprogramação*

O nível de multiprogramação de um sistema não é necessariamente o fator limitante do tamanho máximo da fila de pedidos ("L"); quando o Sistema Operacional divide um programa em múltiplas tarefas ou atividades, e tais tarefas podem executar operações de E/S de forma assíncrona, o número de pedidos é grandemente aumentado, o que resulta em filas de pedidos bem maiores. O nível de multiprogramação poderá ser usado como limitante superior no comprimento da fila de pedidos para cada um dos dispositivos quando vários dispositivos forem utilizados, pois neste caso, a carga de trabalho será menor para o conjunto como um todo.

c) *Uso de múltiplos subsistemas de disco*

As configurações que utilizam múltiplos dispositivos afetam o desempenho do subsistema de disco de duas formas principais. Em primeiro lugar, se os pedidos forem considerados uniformemente distribuídos entre os dispositivos, a demanda para cada dispositivo individualmente será bastante reduzida. Em segundo lugar, muitos dispositivos podem estar sendo servidos por um único canal de E/S. Neste caso, a causa do baixo desempenho pode estar sendo a saturação do controlador ou do canal de E/S, não importando a eficiência do acionador de disco. Portanto, deve-se considerar cuidadosamente os efeitos do escalonamento nesses casos.

d) *Distribuição de pedidos não-uniforme*

A distribuição de pedidos em qualquer sistema é altamente dependente da carga de trabalho e não pode ser generalizada. Algumas causas de distribuições não-uniformes são o uso de dispositivos físicos dedicados a um único programa (exemplo: discos removíveis), o emprego de prioridades para uso dos discos e a alocação de arquivos muito utilizados, ou diretórios em alguns cilindros contíguos, normalmente próximos do centro do disco. Estas técnicas tendem a reduzir a

eficiência dos métodos de escalonamento e, em alguns casos, podem ser usadas no lugar deles. Além disto, se alguns cilindros contiverem muitos pedidos, métodos de otimização rotacional (Seção 2.7.5) poderão ser utilizados em conjunto com o escalonamento para melhorar o desempenho.

e) *Métodos de organização dos arquivos*

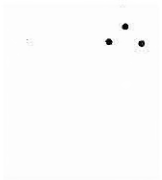
Muitos programas disponíveis comercialmente utilizam vários tipos de organização de arquivos: sequencial, direta ("hashing"), sequencial indexada etc. A característica comum de todas as técnicas é que elas requerem múltiplos acessos ao disco para a obtenção de cada bloco de dados. O método de acesso sequencial indexado ISAM precisa de um acesso para o índice mestre, para o índice do cilindro e para o bloco de dados propriamente dito. Se todos os acessos ao disco para obter um único bloco forem considerados como pedidos consecutivos, o controle do braço do disco será mantido até que o bloco seja acessado, ou seja, no pior caso, até três acessos aleatórios consecutivos poderiam ser necessários para obter um único bloco. No entanto, normalmente o índice mestre fica localizado na memória principal, sendo necessários apenas dois acessos ao disco. Em certas condições, o índice do cilindro pode também ficar localizado na memória principal e, então, a busca do bloco fica reduzida a um único acesso ao disco, mas uma maior parte da memória principal é gasta com os arquivos de índices.

f) *Tempo de procura não-dominante*

Outro fator que diminui o efeito do escalonamento é a reduzida razão entre o tempo de procura ("seek time") e o tempo de acesso total. Verifica-se que o escalonamento dos movimentos do braço do disco é simplesmente uma maneira de reduzir o tempo de procura pela trilha desejada; isto só terá um efeito significativo no tempo total, caso o tempo de procura seja o fator dominante. Um limite superior para este fator dominante é estabelecido pelas características físicas

do dispositivo utilizado. Alguns dispositivos são limitados por um tempo de rotação muito longo como é o caso dos dispositivos flexíveis. Outras reduções na dominação do tempo de procura são causadas por operações múltiplas, tais como "escreva e verifique", repetições devidas a erros de leitura/escrita (problemas de "hardware") e atrasos devidos a canais de E/S ocupados.

Concluindo, apesar da grande quantidade de pesquisas sobre o escalonamento dos movimentos do braço dos discos, verifica-se que em muitas circunstâncias não é necessário o escalonamento, pois, de acordo com Lynch (1972), as filas de pedidos são normalmente pequenas (um ou mais pedidos) e o braço raramente precisa se mover, uma vez que a maioria dos discos possuem um único arquivo aberto num dado instante, e este arquivo é normalmente alocado e acessado sequencialmente. Portanto, em muitos casos, todos os algoritmos de escalonamento são efetivamente equivalentes e qualquer deles, mesmo o FCFS, é uma escolha razoável.



CAPÍTULO 3

PROPOSTA DE UM MONITOR DE DISCO DE ALTO DESEMPENHO PARA APLICAÇÕES EM TEMPO REAL

3.1 - INTRODUÇÃO

De posse dos conceitos básicos apresentados no Capítulo 2, apresenta-se aqui uma proposta para a implementação de um Monitor de Disco de Alto Desempenho.

Em primeiro lugar, é apresentada a estrutura geral adotada, ressaltando a estrutura lógica criada pelo monitor e seu mapeamento na estrutura física.

Em seguida, é feita uma descrição dos níveis da estrutura, de suas funções e das interações entre esses níveis.

Apresenta-se ainda o conjunto de alternativas e técnicas que compõem o Monitor de Disco, de modo a atender aos objetivos citados no Capítulo 1. Propõe-se o conjunto de primitivas do monitor.

3.2 - ESTRUTURAÇÃO GERAL DO MONITOR

Como citado no Capítulo 1, a estrutura lógica criada pelo Monitor de Disco fornece aos processos de aplicação uma visão lógica composta de diretórios associados aos discos magnéticos, os quais contém arquivos que são formados por registros lógicos acessíveis através do monitor, como mostra a Figura 3.1.

Através dessa estrutura lógica, os processos podem manipular os discos e seus arquivos de uma forma lógica, através de nomes simbólicos, não se preocupando com as características físicas dos periféricos utilizados

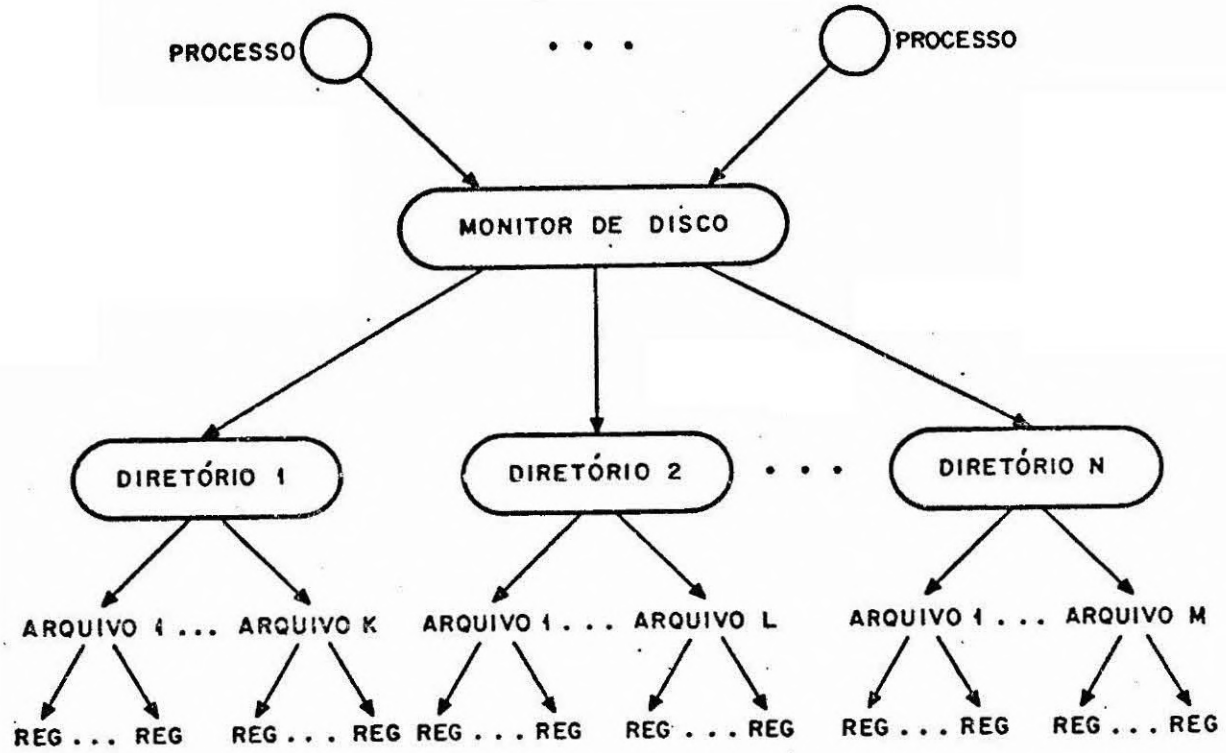


Fig. 3.1 - Estrutura lógica criada pelo monitor.

A estrutura física para qual deve ser mapeada a estrutura lógica descrita é apresentada na Figura 3.2.

Compõe-se dos discos magnéticos divididos em trilhas e estas, em setores físicos. Os controladores dos discos fazem a interface entre o acionador e as rotinas internas da máquina utilizada. A relação entre os processos e o disco magnético é mostrada na Figura 3.3, realçando a estrutura lógica e o seu mapeamento na estrutura física.

Para realizar o mapeamento da estrutura lógica para a física, o monitor implementa vários procedimentos que irão permitir aos processos de aplicação a utilização do disco magnético de maneira eficiente. Esses procedimentos, chamados "primitivas do monitor", são apresentadas na Seção 3.4.7 neste capítulo.

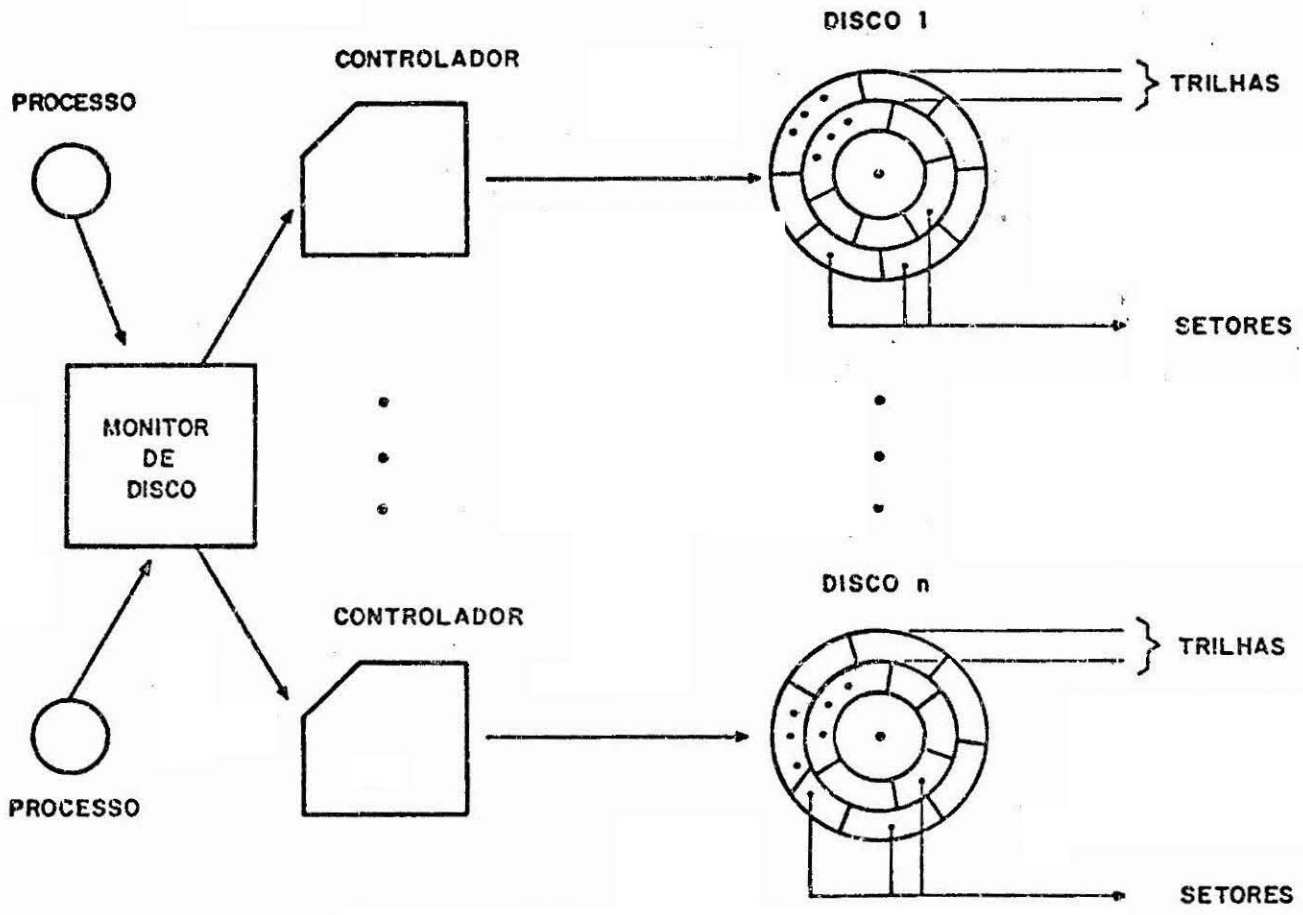


Fig. 3.2 - Estrutura física mapeada pelo monitor.

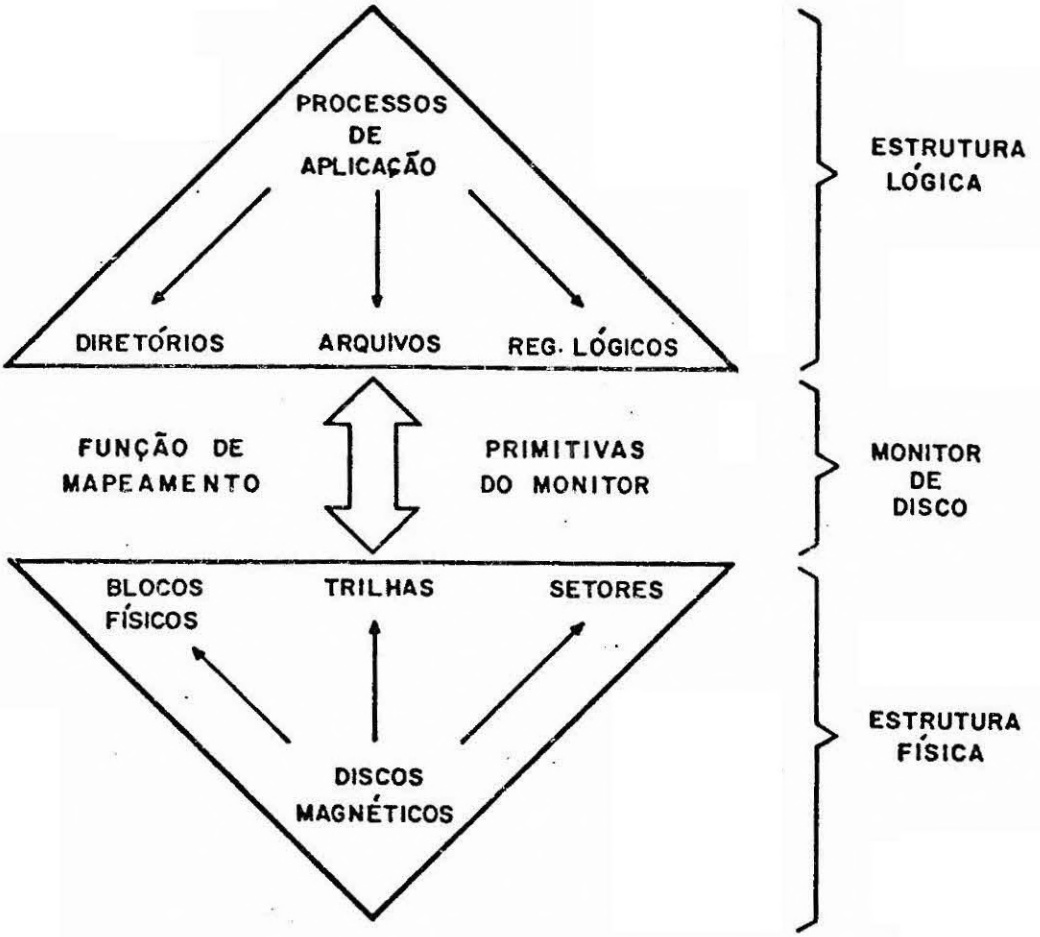


Fig. 3.3 - Função de mapeamento do monitor.

Do acima exposto, conclui-se que os processos de aplicação vão armazenar as informações em entidades lógicas que são os arquivos do sistema. Tais arquivos são compostos de vários registros lógicos, os quais são a unidade mínima de acesso aos arquivos por parte dos processos.

Os arquivos podem ser manipulados através de nomes simbólicos a eles relacionados e estão organizados em diretórios associados a cada dispositivo físico (discos magnéticos).

Para permitir a utilização eficiente dos discos magnéticos, o acesso a eles deve adequar-se às características físicas particulares de cada um. Para os discos, a unidade de acesso é conhecida como um setor físico, o qual pode ser composto de um ou mais registros lógicos, dependendo do tipo de disco e do controlador.

Para os dispositivos de armazenamento rotacionais, como é o caso dos discos magnéticos, a maneira mais eficiente de manipular as informações é dispô-las em grupos de setores contíguos, o que permite que as informações sejam acessadas mais rapidamente.

Tais grupos são chamados "blocos físicos", os quais vão ser a unidade de acesso por parte dos algoritmos que compõem o monitor, visando conseguir o máximo desempenho no armazenamento e na recuperação das informações.

As várias unidades de acesso ao disco são definidas para se adaptarem melhor aos seguintes fatores:

- características do disco e contralador utilizados,
- padrões de acesso dependentes da aplicação.

A definição destes parâmetros é de vital importância no desempenho do monitor e pode ser feita para cada tipo de aplicação específica. Não existe uma relação fixa entre esses parâmetros. Geralmente, existe uma relação típica adotada em vários sistemas citados em Hogam (1983), Mckeon (1985) e Ritchie e Thompson (1984), que é:

bloco físico > setor físico > registro lógico.

A Tabela 3.1 mostra o relacionamento entre as unidades de acesso acima citada e seus tamanhos típicos, classificadas de acordo com o nível que faz o acesso.

TABELA 3.1

BLOCAGEM DO MONITOR

Acesso Via	Blocagem	Tamanho Típico
Processo (Aplicação)	Registro Lógico	128 - 1024 "Bytes"
Algoritmos do monitor	Bloco Físico	5 - 10 Setores Físicos
Controlador (hardware)	Setor Físico	1 - 4 Registros lógicos

Como o Monitor de Disco possui uma estrutura complexa, sua implementação deve ser feita de maneira estruturada.

Isso pode ser obtido com a adoção da estrutura hierárquica, composta de vários níveis, apresentada no Capítulo 1. Tal estrutura, mostrada na Figura 1.1, possui várias vantagens de acordo com Madnick e Aslop (1969):

- *Completeza lógica:* cada nível possui funções bem definidas. As interações entre os vários níveis são mais claras, o que facilita a implementação modular.

- *Facilidade de manutenção (depuração):* os erros podem ser localizados e isolados em um dado nível, o que facilita sua identificação. Isto seria muito difícil caso as interações entre os módulos não fossem bem definidas.

- *Facilidade de modificação (aperfeiçoamento ou expansão):* as modificações que se fizerem necessárias irão afetar apenas um ou dois níveis de estrutura, o que facilita a adaptação do monitor a ambientes físicos distintos, aumentando a portabilidade do monitor.

Entretanto, deve-se observar que a estrutura hierárquica pode prejudicar o desempenho do monitor, uma vez que existem várias interfaces entre os níveis, as quais devem ser obedecidas por todas as primitivas de acesso. Existe um compromisso entre o desempenho e a modularidade que deve ser considerado para cada aplicação.

Adotou-se aqui tal estrutura, pois as vantagens oferecidas são maiores que as desvantagens e procurou-se utilizar soluções que sejam simples e maximizem o desempenho como um todo. Na Seção 3.3 faz-se o detalhamento da estrutura adotada.

Para que o monitor seja versátil, ele deve possuir várias ferramentas de manipulação de arquivos que possam ser utilizadas pelos processos em tempo de execução; para que ele se mantenha simples e efeciente, vários parâmetros de operação devem ser fixos para cada aplicação. Assim, propõe-se que o Monitor de Disco possua duas fases distintas de operação:

1) *Fase de configuração:* nesta fase são estabelecidos os vários parâmetros de operação do monitor, que são fixos para cada aplicação. São também criados os diretórios nos dispositivos a serem controlados pelo monitor. Essas operações são realizadas pelo usuário através de "primitivas de configuração".

2) *Fase de execução:* após a fase de configuração, os processos de aplicação podem passar a operar com os arquivos através de "primitivas de manipulação". Os processos não têm acesso às primitivas de configuração. Isto assegura a integridade do Monitor de Disco.

A Figura 3.4 mostra o fluxograma de execução do monitor, salientando as fases de configuração e de execução.

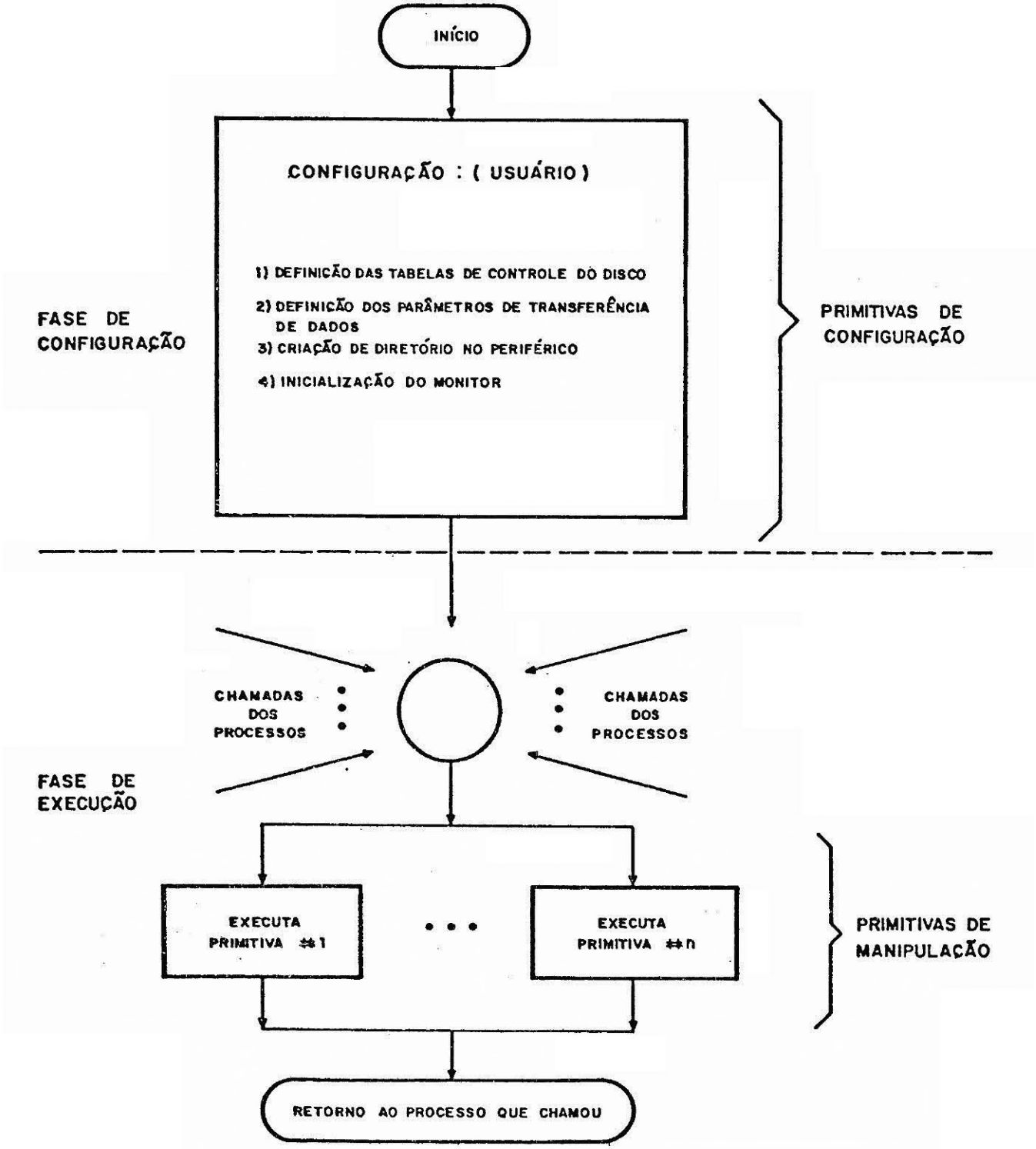


Fig. 3.4 - Fluxograma de execução do monitor.

A operação do monitor é iniciada pela fase de configuração. O código do monitor é carregado pelo usuário que vai fazer a configuração para uma determinada aplicação. Através das primitivas de configuração, o usuário define todos os parâmetros e cria os diretórios nos discos desejados, montando o ambiente de execução. Em seguida, é chamada a primitiva de inicialização que finaliza a fase de configuração e inicia a de execução.

A partir daí os processos de aplicação podem iniciar a operação normal com os arquivos, utilizando as primitivas de manipulação disponíveis.

3.3 - DETALHAMENTO DE ESTRUTURA HIERÁRQUICA

A estrutura hierárquica adotada para a implementação do Monitor de Disco é mostrado ha Figura 1.1. A seguir, apresenta-se o detalhamento das funções implementadas pelos vários níveis desta estrutura.

3.3.1 - SISTEMA LÓGICO DE ARQUIVOS (NÍVEL 5)

O Sistema Lógico de Arquivos (S.L.A) faz a interface entre os processos de aplicação e o Monitor de Disco através das primitivas implementadas no nível 5.

O S.L.A associa os nomes simbólicos aos arquivos, utilizando para isto o diretório. Em seguida, determina-se um identificador para o arquivo, utilizado para acessar o seu descritor. O descritor contém todas as informações necessárias para a localização, proteção e manutenção do arquivo. Este mecanismo é mostrado na Figura 3.5.

As referências aos arquivos por parte dos processos de aplicação são feitas através de nomes simbólicos, os quais não dependem das características físicas dos meios de armazenamento. Deste modo, isolam-se os processos dos dispositivos físicos.

As primitivas implementadas pelo S.L.A são a de criação e remoção de arquivos, apresentada na Seção 3.4.7.

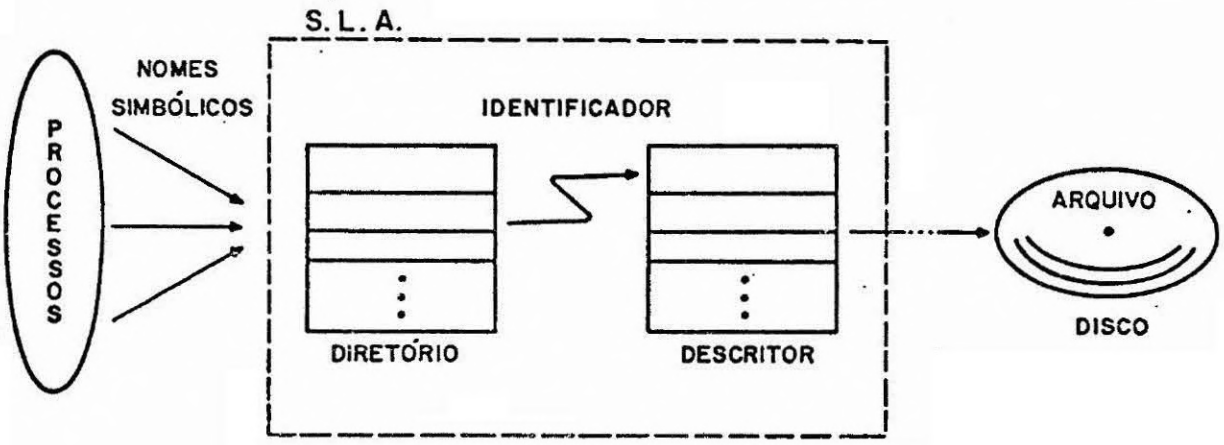


Fig. 3.5 - Função do S.L.A.

3.3.2 - SISTEMA BÁSICO DE ARQUIVOS (NÍVEL 4)

As principais funções do Sistema Básico de Arquivos (S.B.A) são:

- busca de descritores dos arquivos,
- proteção dos acessos aos arquivos,
- identificação dos registros lógicos pertencentes aos arquivos,
- atualização e manutenção dos descritores.

Através do identificador único do arquivo fornecido pelo S.L.A. o S.B.A deve buscar o descritor do arquivo no disco e passá-lo para a memória principal, caso este arquivo ainda não esteja em uso.

Como as informações de proteção do acesso ao arquivo ficam no descritor, o S.B.A deve utilizá-las para fazer a validação do tipo de acesso pretendido para este arquivo.

Através do identificador único do arquivo fornecido pelo S.L.A. o S.B.A deve buscar o descritor do arquivo no disco e passá-lo para a memória principal, caso este arquivo ainda não esteja em uso.

Como as informações de proteção do acesso ao arquivo ficam no descritor, o S.B.A deve utilizá-las para fazer a validação do tipo de acesso pretendido para este arquivo.

O descritor contém também informações sobre como identificar os registros lógicos que pertencem a um determinado arquivo. Estas informações são passadas ao próximo nível da hierarquia que faz o mapeamento entre os registros lógicos e os setores físicos.

O S.B.A deve providenciar também a atualização dos descritores dos arquivos contidos no disco, após as alterações por ele efetuadas. Isto é necessário para manter a consistência entre as informações contidas no disco e as contidas na memória.

As primitivas implementadas pelo S.B.A são as que trabalham com os descritores dos arquivos, como as de abertura e fechamento de arquivos e as de manipulação dos descritores. Tais primitivas são propostas na Seção 3.4 e detalhadas no próximo capítulo.

3.3.3 - SISTEMA DE ORGANIZAÇÃO DE ARQUIVOS (NÍVEL 3)

As funções do Sistema de Organização de Arquivos (S.O.A.) são:

- mapeamento lógico para o físico,
- alocação do espaço do disco.

O S.O.A faz o mapeamento dos pedidos de acesso aos registros lógicos recebidos dos níveis superiores para os pedidos de acesso aos setores físicos, os quais são enviados ao próximo nível da estrutura que irá acessar o disco magnético.

Esse mapeamento é fortemente dependente das características físicas do disco utilizado e do método de alocação do espaço adotado. O método de mapeamento a ser adotado tem grande influência no desempenho do monitor. Este deve ser feito de maneira simples, de modo que minimize os cálculos para a obtenção dos setores físicos dos arquivos.

Os vários parâmetros necessários à realização do mapeamento são específicos para cada aplicação e dependem das características físicas do "hardware" adotado. Tais parâmetros podem ser definidos pelo usuário na fase inicial, através de uma primitiva de configuração a ser apresentada na próxima seção. Esta primitiva cria tabelas e armazena os parâmetros para uso na fase de execução.

A alocação do espaço do disco deve atender a dois requisitos:

- permitir um rápido acesso às informações armazenadas,
- conseguir um bom aproveitamento do espaço do disco.

Para as aplicações em tempo real, geralmente é mais importante conseguir a minimização do tempo de acesso do que o aproveitamento de espaço livre que não é primordial. A principal técnica para minimizar o tempo com a alocação consiste em alocar vários blocos físicos de cada vez, o que permite que, uma vez encontrado um bloco, os outros sejam automaticamente encontrados. Este aspecto será levado em conta na hora da seleção do método de alocação a ser abordado na Seção 3.4.4.

3.3.4 - CONTROLE DE ENTRADA E SAÍDA (NÍVEL 2)

O Controle de Entrada e Saída (C.E.S) tem a função principal de implementar algoritmos de maximização do desempenho do monitor. Neste nível existe a preocupação com dois aspectos de otimização:

- minimização dos movimentos da cabeça móvel do acionador de discos, o que reduz o tempo de acesso (escalonamento);
- otimização das transferências de dados entre os periféricos e a memória principal.

Até agora a ordem dos pedidos de acesso aos arquivos era determinada pelos processos requisitantes. O C.E.S, visando o aproveitamento das características físicas dos dispositivos e a minimização do tempo de acesso aos dados, altera a ordem dos pedidos recebidos do S.B.A, de modo a passá-los ao controlador numa sequência ótima que permita obter um maior desempenho do disco magnético.

A otimização das transferências de dados entre o periférico e a memória é obtida aqui com o uso de regiões de memória, chamadas "buffers", que são utilizados para minimizar o número de acessos ao disco. Como os acessos ao disco são operações lentas, sua minimização contribui para o aumento do desempenho do monitor.

As primitivas aqui implementadas são as de acesso aos setores físicos do disco, tais como leitura e escrita do setor físico. As técnicas de transferência de dados e de escalonamento são apresentadas na Seção 3.4.6.

3.4 - SELEÇÃO DE ALTERNATIVAS E TÉCNICAS

A partir do exposto nas seções anteriores, foi elaborada a Tabela 3.2, que mostra a estrutura geral do monitor, relacionando os vários níveis, suas funções principais, as estruturas de dados afetadas e a unidade de acesso utilizada em cada um dos níveis. Pode-se observar pela tabela que os níveis 4 e 5 implementam a estrutura lógica criada pelo Monitor de Disco, a qual é mapeada para a estrutura física implementada pelos níveis 2 e 3 da hierarquia.

De posse dessas informações e dos objetivos a serem atingidos é feita a seleção das alternativas de implementação a serem adotadas com base nos conceitos teóricos apresentados no Capítulo 2. A sequência dos tópicos a ser adotada aqui é:

- organização dos arquivos,
- sistema de diretório,
- proteção dos arquivos,
- alocação do espaço no disco,
- controle do espaço livre,
- métodos específicos para otimização,
- primitivas do Monitor.

TABELA 3.2

ESTRUTURA GERAL DO MONITOR

	NÍVEIS	FUNÇÕES PRINCIPAIS	ESTRUTURAS DE DADOS AFETADAS	UNIDADE DE ACESSO
LÓGICO	S.L.A (nível 5)	- associação de nomes simbólicos à identificadores	diretórios	arquivos
	S.B.A (nível 4)	- manipulação dos descritores, - identificação dos registros do arquivo	descritores dos arquivos	registros lógicos
FÍSICO	S.O.A (nível 3)	- mapeamento lógico físico, - alocação do espaço	- tabelas de parâmetros dos dispositivos - tabelas de alocação	blocos físicos
	C.E.S (nível 2)	- otimização das transferências de dados, - escalonamento dos pedidos	"buffers" de dados	setores e trilhas

3.4.1 - MÉTODO DE ORGANIZAÇÃO DOS ARQUIVOS

A organização física dos arquivos num sistema de gerenciamento de arquivos é determinada por dois aspectos principais:

- *tipo da aplicação:* cada aplicação exige um determinado padrão de acesso aos arquivos.

- *características dos meios de armazenamento:* os diferentes dispositivos físicos têm diferentes características, o que faz com que as organizações mais eficientes sejam distintas para cada tipo de dispositivo.

Embora as aplicações de tempo real sejam dedicadas, podem existir várias organizações possíveis para os diferentes tipos de arquivos manipulados. Para que o monitor seja utilizado para acesso a todos os tipos de arquivos, ele deve implementar rotinas de acesso específicas a cada tipo de organização, o que aumenta sua complexidade e prejudica o desempenho.

A alternativa proposta, que parece mais adequada aos objetivos do trabalho, é fazer com que o monitor não suporte nenhum tipo de organização para os arquivos de sistema. Os arquivos são tratados sempre como uma sequência de "bytes" agrupados em registros lógicos, setores e blocos. Toda a implementação das organizações necessárias é feita pelo processo de aplicação que utiliza o arquivo.

Essa solução, também utilizada no sistema operacional UNIX (Ritchie and Thompson, 1974), permite uniformizar as rotinas de acesso aos arquivos, o que simplifica o monitor, melhorando seu desempenho.

Outra vantagem é que o monitor fica transparente ao tipo de organização do arquivo, o que aumenta sua portabilidade (Peterson and Silberschatz, 1983),

Uma pequena restrição desse método é que o sistema fica com menos recursos para o tratamento de erros cometidos pelo usuário. Um exemplo dessa situação é quando se tenta executar um programa fonte qualquer. Para o monitor não importa que o arquivo não seja executável, entretanto sua execução pode provavelmente causar um desastre na máquina.

3.4.2 - SISTEMA DE DIRETÓRIO

O sistema de diretório permite que os arquivos sejam manipulados de forma lógica pelos processos de aplicação, o que aumenta a versatilidade do monitor, pois facilita o acesso aos arquivos. Como as operações relacionadas com o diretório (criação, remoção, abertura e fechamento de arquivos) são, em geral, realizadas antes ou ao final das operações de acesso (escrita e leitura) propriamente ditas, sua influência no desempenho é diminuída. Entretanto, no acesso concorrente a vários arquivos, tais operações podem prejudicar a eficiência do monitor.

Como se deseja que o monitor possua uma estrutura lógica para acesso aos arquivos, propõe-se que exista um sistema de diretório bem simples, implementado com um único nível. Isto implica a existência de apenas um arquivo diretório que contém todos os arquivos do monitor, como mostrado na Figura 2.5. Existe um diretório para cada dispositivo (disco).

As principais vantagens desta estrutura são:

- *simplicidade:* facilita a implementação e contribui para aumentar o desempenho,

- *manutenção:* as alterações desta estrutura são de fácil realização,

- *adequado:* as aplicações onde o número de arquivos não é muito grande, como é o caso das aplicações mais dedicadas.

As principais restrições a esta estrutura são:

- os arquivos devem possuir nomes diferentes, pois estão todos num único diretório,
- menor versatilidade, o que impossibilita a criação de subdiretórios dedicados a cada processo.

Os diretórios do tipo árvore e do tipo grafo (simples ou acíclico) são mais versáteis, possibilitam o compartilhamento de arquivos e suportam arquivos com nomes idênticos desde que em diretórios distintos. A grande desvantagem destes sistemas é a complexidade dos algoritmos de acesso e o maior tempo exigido para percorrer tais estruturas.

Conclui-se então que é mais vantajosa a adoção de uma estrutura simples, que forneça poucas facilidades de acesso, mas que não prejudique desnecessariamente o desempenho do monitor.

Como citado anteriormente, os diretórios são criados por uma primitiva de configuração durante a fase de geração do sistema. Também nesta fase são fixados o número total de arquivos do diretório e o número máximo de arquivos em uso num mesmo instante. Estes parâmetros são característicos de cada configuração.

3.4.3 - MÉTODO DE PROTEÇÃO DOS ARQUIVOS

Como nos sistemas de tempo real o número de processos ativos não é grande em relação aos sistemas multiusuários e os acessos aos arquivos podem ser bastante controlados, não existe a necessidade de implementar um método de proteção complexo, pois isto iria prejudicar o desempenho sem trazer um ganho real.

Após a análise dos métodos de proteção disponíveis, propõe-se que seja adotado um esquema de proteção que atenda aos seguintes objetivos:

- *proteção contra acesso indevido:* os processos que não possuam permissão não podem acessar o arquivo,

- *proteção ao acesso concorrente:* deve-se evitar que dois processos acessem simultaneamente um mesmo arquivo durante as operações de escrita,

- *proteção seletiva:* o acesso aos arquivos pode ser seletivo, isto é, apenas para leitura, apenas para escrita, apenas para execução ou uma combinação destes três tipos.

A proteção contra o acesso indevido é conseguida associando a cada arquivo uma lista com os nomes dos processos e o tipo de acesso permitido ao arquivo. Esta lista de proteção tem um tamanho fixo determinado na fase de configuração, enquanto seu conteúdo é fornecido na hora da criação do arquivo.

A proteção ao acesso concorrente durante a alteração do arquivo é realizada utilizando um indicador de operação de escrita associado ao arquivo e aramzenado no descritor. Um dado registro de um arquivo só pode ser acessado concorrentemente por mais de um processo se ele não estiver sendo modificado por nenhum destes processos.

A proteção seletiva é conseguida associando a cada processo da lista de acesso do arquivo um campo que indique qual o tipo de acesso permitido. Esta informação também é fornecida na criação do arquivo.

O uso de listas de acesso associadas a cada arquivo é possível quando o número de processos não é grande, como é o caso das aplicações em tempo real para as quais o monitor é proposto. Esta solução é de implementação simples e ainda permite o acesso seletivo aos arquivos, não comprometendo o desempenho do monitor.

3.4.4 - MÉTODO DE ALOCAÇÃO DE ESPAÇO DO DISCO

O método de alocação é um dos fatores que exercem grande influência no desempenho. Para que o método seja eficiente, ele deve con

siderar as características físicas dos meios de armazenamento. No caso dos discos magnéticos, a alocação é mais eficiente quando é feita em blocos de tamanho fixo (veja Seção 2.6).

O tamanho do bloco físico é de grande importância na eficiência do método de alocação e depende de vários fatores, tais como do tipo de acesso e do "hardware" utilizado.

Na Seção 2.7.1 discutiu-se o problema do uso de blocos pequenos e sugere-se o uso de blocos maiores, mesmo com o problema da fragmentação interna que pode ocorrer. A definição do tamanho do bloco é fixa e feita na geração do sistema através de uma primitiva de configuração.

Após a análise dos métodos de alocação existentes (Seção 2.6) e tendo em mente o objetivo principal do trabalho e sua aplicação básica, propõe-se a adoção do método de *alocação contínua* para os arquivos do sistema. As razões desta medida são basicamente as seguintes:

- *Facilita* o armazenamento de arquivos de forma sequencial mais rapidamente. Isto é importante para o armazenamento de dados provenientes de processos de aquisição serial de alta velocidade.
- Intrinsicamente, prove uma *minimização* do número de movimentos da cabeça do disco necessários para o acesso aos vários registros de um arquivo.
- Os procedimentos de acesso ao arquivo são mais *simples* de ser implementados.
- Permite o *acesso não-sequencial*, quando necessário, na recuperação dos arquivos.

O grande problema que existe com essa solução é o da fragmentação externa do espaço livre do disco. Entretanto, este é um compromisso a ser assumido e pode ser minimizado com a utilização de procedimentos de compactação executados em intervalos tais que não comprometam a operação do sistema. Estes procedimentos são programas utilitários a serem desenvolvidos utilizando as primitivas do monitor.

O método de procura do espaço livre para a alocação dos arquivos é o método conhecido como "first-fit"; ou seja, e a utilização do primeiro espaço de tamanho suficiente que for encontrado. Esta escolha é justificada por ser este método mais rápido que o "best-fit" e de eficiência equivalente (Shore, 1975).

Para minimizar a fragmentação do espaço, o algoritmo de alocação faz a alocação do espaço em blocos físicos que são compostos de vários setores físicos. Esta técnica também facilita o controle do espaço livre do disco.

Para aumentar o desempenho do monitor, propõe-se a utilização da técnica de *pré-alocação* do espaço, que consite na alocação antecipada de um número adicional de blocos do disco.

Com isso diminui-se sensivelmente o número de acessos ao disco necessários durante a escrita, pois se antecipa a necessidade de espaço para o armazenamento do arquivo. O número de blocos a ser pré-alocado é um parâmetro definido na fase de configuração.

Ainda em relação à alocação do espaço no disco, deve-se considerar o posicionamento dos vários setores no disco. Embora seja utilizada a alocação contínua para os arquivos, seus setores físicos não devem ser localizados em posições adjacentes em uma mesma trilha.

Esse aspecto foi amplamente discutido na Seção 2.7.4 e conclui-se que se pode otimizar muito o acesso ao disco adotando a técnica de *entrelaçamento de setores*. Deve-se ressaltar que o entrelaçamento faz com que o espaço do disco seja então alocado de maneira logicamente contínua, pois, de maneira transparente para o "software", os setores logicamente contíguos não são fisicamente contíguos.

Existem basicamente duas técnicas de entrelaçamento normalmente utilizadas:

- entrelaçamento que utiliza *tabelas de translação* de setores: neste caso, o entrelaçamento é feito em tempo real, quando for necessário acessar um setor físico.

- entrelaçamento feito durante a *formatação* do disco: neste caso, o entrelaçamento é feito por um programa dedicado, antes da utilização do disco.

A primeira técnica, utilizada em sistema CP/M (Hogan, 1983), tem a grande desvantagem de exigir consultas a tabelas de translação a cada acesso ao disco, o que gasta tempo adicional.

No segundo caso, a translação é fixa e transparente para o "software" que "pensa" estar acessando o disco sequencialmente. O próprio controlador irá acessar os setores de maneira entrelaçada, pois a formatação do disco foi feita desta maneira.

Conclui-se que a segunda alternativa é a mais adequada para o monitor proposto. O programa de formatação que vai preparar o disco para ser usado pelo monitor deve estar disponível como um utilitário do sistema.

3.4.5 - MÉTODO DE CONTROLE DO ESPAÇO LIVRE

Das várias técnicas de gerenciamento do espaço livre apresentadas na Seção 2.6.1, a que é normalmente adotada em sistemas de menor porte é a da utilização de um *vetor de bits*, no qual cada bit representa um bloco físico disponível do disco, e o estado deste bit indica se este bloco está livre ou não.

Propõe-se que seja utilizada esta técnica no monitor, pois ela possui as seguintes vantagens:

- *Eficiência:* utiliza apenas um bit para a representação de cada bloco do disco, em vez de uma palavra inteira.

- *Velocidade no acesso:* a consulta torna-se mais rápida, pois a cada acesso ao vetor são lidos vários bits simultaneamente, contendo informações sobre muitos blocos.
- *Economia de espaço:* como a informação é bastante compactada, os vetores de bits ocupam pouco espaço, podendo ser facilmente mantidos na memória principal durante a utilização do disco.

Os detalhes da implementação do método são abordados no próximo capítulo.

3.4.6 - MÉTODOS ESPECÍFICOS DE OTIMIZAÇÃO

Até este instante, preocupou-se em adotar soluções que sejam as mais eficientes e adequadas para os objetivos do trabalho. Todas as soluções visam a otimização de vários aspectos diferentes da implementação.

De acordo com o citado na Seção 2.8.5, o uso de técnicas específicas de otimização do desempenho só é eficiente se vários outros fatores forem observados.

Existem dois aspectos de grande influência no desempenho que ainda não foram abordados até aqui:

- otimização das transferências de dados,
- minimização dos movimentos da cabeça do disco.

1) Otimização das transferências de dados

A técnica normalmente utilizada para a otimização das transferências de dados é o uso de áreas da memória ("buffers") como meio de adequar a velocidade de transferência de dados dos periféricos à velocidade do processador.

Na Seção 2.7.6 foram apresentados os conceitos básicos do uso de "buffers" e as técnicas mais comumente utilizadas.

Na Seção 2.7.7 foi apresentada a técnica de memória "cache", muito utilizada para a otimização das transferências de dados.

Existem dois mecanismos básicos para esta otimização:

- utilização de "buffers" de maneira simples,
- utilização de "buffers" como parte de um sistema de memória "cache".

Em ambos os casos deve-se ressaltar que os arquivos devem ser alocados sequencialmente no disco, a fim de que realmente haja um ganho do desempenho devido ao uso de "buffers".

No primeiro caso, a grande desvantagem decorre do fato de que, quando os "buffers" ficam cheios, seu uso não fornece nenhum ganho adicional.

No segundo caso, o sistema de memória "cache" é mais elaborado e utiliza algoritmos de substituição das informações contidas nos "buffers", de modo a fazer com que as informações mais acessadas sempre estejam presentes nestes. Embora, neste caso, os algoritmos sejam mais complexos, consegue-se minimizar drasticamente o número de acessos ao disco. Como os acessos ao disco são operações muito mais lentas que as operações feitas pela CPU, o ganho do desempenho é bem maior.

Propõe-se que seja utilizada um método de memória "cache" para otimizar as transferências de dados.

Adotou-se como base para este trabalho o método apresentado por Mckeon (1985), cuja característica principal é utilizar um algoritmo de substituição, o qual desloca o "buffer" que possua a menor frequência de acesso e também seja o menor usado recentemente ("Least Recently Used - L.R.U").

O controle desses parâmetros é feito por meio de indicadores ("flags") contidos em um cabeçalho associado a cada "buffer".

Algumas modificações nos algoritmos são introduzidas para melhorar seu desempenho e adequá-lo ao caso do monitor.

Uma modificação básica é a programação de vários parâmetros de operação do algoritmo na fase de configuração, como é o caso do número máximo e do tamanho dos "buffers" do "cache".

O detalhamento do algoritmo é apresentado no próximo capítulo.

2) Minimização dos movimentos da cabeça (escalonamento)

Conforme o exposto na Seção 2.8.5 e apresentado por Teorey (1972), Peterson e Silberchatz (1983) e Smith (1981), vários fatores afetam a eficiência dos algoritmos de escalonamento num sistema real.

Em certos casos, o ganho de desempenho conseguido pelo uso de um algoritmo complexo pode ser anulado por vários fatores, o que justifica o uso de algoritmos mais simples como o "first-come-first-served" (Seção 2.8.1) ou "shortest-seek-time-first" (Seção 2.8.2).

Analisando os métodos de escalonamento mais utilizados descritos na Seção 2.8, conclui-se que para cada tipo de situação deve-se utilizar um método diferente:

a) Na situação em que o escalonamento pode ser eficiente, sem sofrer as influências de fatores externos (Seção 2.8.5), deve-se usar o método C-SCAN que fornece a menor variância do tempo de espera sem prejuízo do tempo de resposta (Seção 2.8.3).

b) Na situação em que os pedidos são de natureza aleatória e o escalonamento torna-se ineficiente, deve-se usar um método simples como o FCFS ("first-como-first-served") apresentado na Seção 2.8.1.

Em Mckeon (1985), são apresentados dois métodos utilizados para acessar o disco, cada um adequado a um tipo de situação:

- *operação de escrita no disco:* as escritas no disco são retardadas até que vários blocos sejam escritos de uma só vez e em ordem, o que minimiza o número de acessos ao disco e os movimentos de cabeça deste. São utilizados vários "buffers", para possibilitar a ordenação dos pedidos de escrita, a fim de que estes sejam realizados sequencialmente no sentido do movimento da cabeça do disco (algoritmo tipo C-SCAN).

- *operações de leitura no disco:* como os pedidos de leitura são de natureza aleatória, estes são enviados ao controlador na ordem de chegada (algoritmo do tipo FCFS). Neste caso, consegue-se minimizar o número de acessos fazendo a leitura antecipada (Seção 2.7.5) dos setores do disco através do uso de "buffers" na memória.

Propõe-se a utilização dos algoritmos apresentados em Mckeon (1985), introduzindo modificações para que seja feita a leitura antecipada de blocos adicionais aproveitando os "buffers" do sistema "cache" usado nas transferências de dados.

As técnicas apresentadas nesta seção são implementadas no nível 2 da hierarquia proposta (C.E.S), onde são feitas as leituras e escritas no disco ao nível de setores físicos.

Os algoritmos supõem a existência de rotinas de E/S ("drivers") no Sistema Operacional. Estas rotinas são especificadas no próximo capítulo e sua implementação foge do escopo deste trabalho.

3.4.7 - PRIMITIVAS DO MONITOR

Como exposto na definição geral do Monitor de Disco, propõe-se que este possua duas classes de primitivas:

1) *Primitivas de Configuração:* utilizadas para a definição de vários parâmetros do monitor, a serem utilizados durante a fase de execução. Tais primitivas são executadas antes do uso do monitor, durante sua inicialização pelo usuário que vai configurá-lo.

2) *Primitivas de Manipulação:* utilizadas pelos processos de aplicação para o manuseio dos arquivos, durante a fase de execução. Elas funcionam de acordo com as definições feitas na fase de configuração.

Isso vai permitir que o Monitor de Disco seja configurado de acordo com as características de cada aplicação e do "hardware" de cada máquina utilizada.

As primitivas ainda podem ser classificadas em dois tipos, de acordo com a sua área de atuação:

a) *externas:* são as primitivas disponíveis aos processos de aplicação para acesso ao monitor.

b) *internas:* são as primitivas utilizadas internamente pelo monitor como interface entre os vários níveis da estrutura; elas atuam nas várias estruturas de dados do monitor.

A elaboração das primitivas foi feita considerando as necessidades das aplicações em sistemas de tempo real, onde é exigido um alto desempenho em troca de maiores facilidades de operação. Apesar disto, o conjunto de primitivas proposto permite a manipulação lógica dos arquivos do monitor. Como base foi adotado o padrão MOSI (IEEE, 1984) citado na Secão 2.4.1 (Tabela 2.1).

A implementação das primitivas é realizada pelos vários níveis da estrutura hierárquica de acordo com a função, e as estruturas de dados acessadas por elas. As Tabelas 3.3 e 3.4 apresentam as primitivas propostas.

TABELA 3.3

PRIMITIVAS DE CONFIGURAÇÃO

NOME	DESCRIÇÃO
cria-dir	cria o diretório
def-tcd	define a tabela de controle do disco
def-ptd	define os parâmetros de transferência de dados
inic-MD	inicializa o Monitor de Disco

TABELA 3.4

PRIMITIVAS DE MANIPULAÇÃO DE ARQUIVOS

S.L.A (nível 5)	S.B.A. (nível 4)	S.O.A (nível 3)	C.E.S (nível 2)
cria-arq	abre-arq	leit-reg	leit-set
rem-arq	fecha-arq	escr-reg	escr-set
	monta-descr		
	alt-descr		

A Tabela 3.5 relaciona todas as primitivas do monitor, juntamente com as respectivas funções, os tipos e as classes.

A descrição detalhada das primitvas, suas funções, seus parâmetros e suas estruturas de dados afetadas é apresentada no próximo capítulo.

TABELA 3.5

PRIMITIVAS DO MONITOR

NOME	FUNÇÃO	TIPO	CLASSE
cria-dir	cria diretório no disco	externa	configuração
def-tcd	define tabela de controle do disco	externa	configuração
def-ptd	define parâmetros de transferência de dados	externa	configuração
inic-MD	inicializa Monitor de Disco	externa	configuração
cria-arq	cria novo arquivo	externa	manipulação
rem-arq	remove arquivo do diretório	externa	manipulação
abre-arq	abre arquivo para E/S	externa	manipulação
fecha-arq	fecha arquivo	externa	manipulação
monta-descr	monta descritor na memória	interna	manipulação
alt-descr	altera informações do descritor	interna	manipulação
leit-reg	leitura de registro lógico	externa	manipulação
escr-reg	escrita de registro lógico	externa	manipulação
leit-set	leitura de setor físico	interna	manipulação
escr-set	escrita de setor físico	interna	manipulação

As primitivas de manipulação de arquivos interagem entre si, com os processos e com o núcleo do sistema operacional da máquina, como mostrado no diagrama da Figura 3.6. Para maior clareza do diagrama são mostradas apenas as interações mais relevantes. As primitivas "escr-reg" e "leit-reg" são chamadas para todos os acessos ao disco.

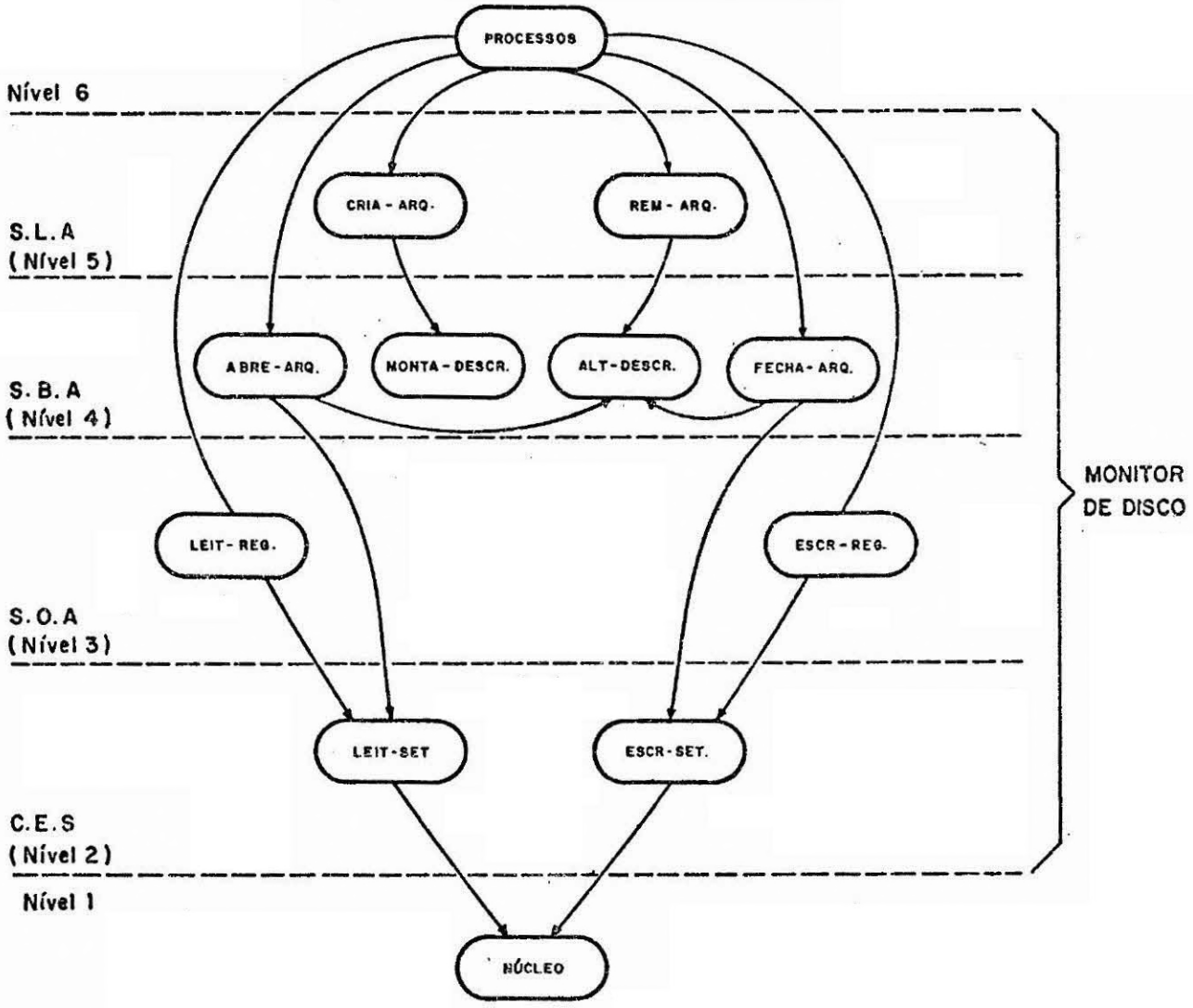


Fig. 3.6 - Interações entre as primitivas de manipulação.

CAPÍTULO 4

DETALHAMENTO DO MONITOR

4.1 - INTRODUÇÃO

Neste capítulo descreve-se detalhadamente a proposta do Monitor de Disco.

Inicia-se pelo detalhamento das soluções adotadas, salientando os algoritmos utilizados e as estruturas de dados necessárias. Mostra-se assim os vários relacionamentos entre os diversos níveis da estrutura e entre os processos, Monitor e dispositivos físicos.

Em seguida, apresentam-se as primitivas selecionadas, mostrando suas funções, cada parâmetro com sua função e um esboço dos fatores influenciados pela escolha dos parâmetros. Mostram-se as primitivas de configuração seguidas pelas de manipulação de arquivos.

Especifica-se também o interfaceamento do monitor com o núcleo de um Sistema Operacional. Mostram-se as considerações aceitas, o modo de implementação a ser adotado e os recursos do núcleo necessários.

4.2 - DETALHAMENTO DO SISTEMA DE DIRETÓRIO

Como citado na Secção 3.4.2, o diretório possui uma estrutura de um único nível. O arquivo diretório é especial no sentido de que ele só pode ser acessado por rotinas do próprio monitor, o que garante a integridade das informações nele contidas.

O arquivo diretório fica armazenado no disco, localizado na trilha 0, ou mais externa, de modo que o restante do espaço do disco possa ser ocupado somente com os arquivos do sistema. Na memória existe uma imagem desse arquivo. Toda vez que um arquivo é criado ou aberto, sua entrada é colocada na imagem do diretório, indicando que ele está

ativo. Na memória só existem as entradas dos arquivos ativos a cada instante.

Um arquivo diretório é criado num disco através da primitiva "cria-dir" durante a inicialização do sistema. Nesta hora são definidos o número máximo de arquivos e o número máximo de arquivos ativos ao mesmo tempo. A primitiva "cria-dir" é detalhada mais adiante no item 3 da Seção 4.6.1.

Além dos nomes dos arquivos, o diretório agrupa outras informações, as quais são então acessadas de uma só vez ao ser lido o diretório do disco. A estrutura do diretório pode ser vista na Figura 4.1.

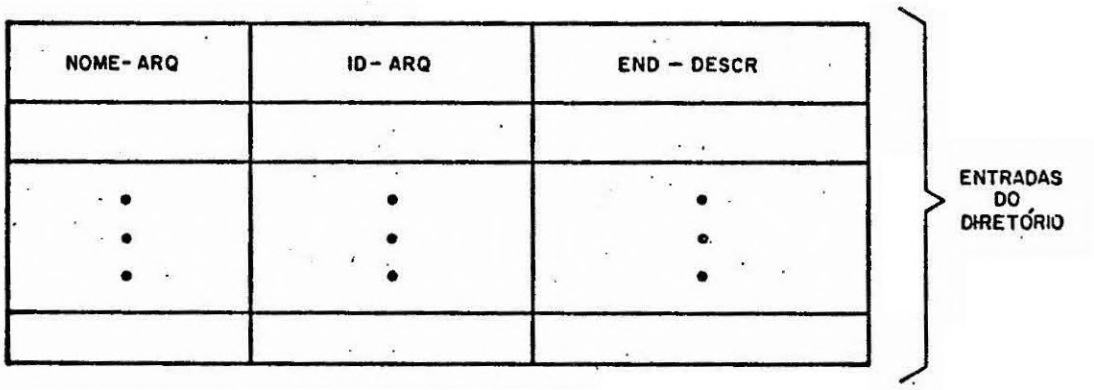


Fig. 4.1 - Arquivo diretório.

A seguir são descritas as informações contidas em cada entrada do diretório:

nome-arq = nome do arquivo de acordo com a convenção adotada pelo Sistema Operacional da máquina em uso.

id-arq = identificador único do arquivo, determinado na sua criação. É um inteiro utilizado para futuras referências ao arquivo por parte das primitivas internas do monitor.

end-descr = endereço do descritor do arquivo. Usado para a localização do descritor.

4.3 - DETALHAMENTO DOS DESCRITORES DOS ARQUIVOS

Os descritores contêm todas as informações necessárias para a manipulação dos arquivos. São tabelas utilizadas pelo S.B.A., as quais residem no disco, junto com o arquivo diretório. Como no caso do diretório, o descritor de um arquivo é trazido para a memória quando este é aberto. Existe uma área na memória para conter todos os descritores dos arquivos ativos num dado instante. Esta área é criada pela primitiva "cria-dir".

Todas as operações efetuadas em um arquivo são registradas por alterações no seu descritor, o qual é reescrito no disco logo após o arquivo ser fechado. O descritor possui um tamanho fixo e contém as seguintes informações:

1) Identificador do arquivo.

2) Nome do processo criador do arquivo.

3) Ponteiro para lista de acesso ou indicador de arquivo público.

4) Indicador de posição para operações de escrita.

5) Indicador de posição para operações de leitura.

6) Ponteiro para tabela de registros lógicos do arquivo.

7) Contador de processos que estão acessando o arquivo e indicador do tipo de operação corrente (escrita ou leitura).

8) Contador de blocos pré-alocados.

9) Contador de blocos do arquivo.

10) Data da criação (opcional).

A descrição detalhada de cada item é feita a seguir:

1) Identificador do arquivo: é um inteiro atribuído a cada arquivo quando ele é criado. Este número é associado ao nome do arquivo e utilizado pelos níveis mais baixos para referir-se ao arquivo.

2) Nome do processo criador: é utilizado para controlar o acesso ao arquivo.

3) Ponteiro que indica o início da lista de acesso, a qual contém o nome de todos os processos que podem acessar esse arquivo, juntamente com o tipo de acesso permitido. Se for igual a zero, indica que o arquivo é público.

4) Inteiro que indica qual será o próximo registro lógico a ser acessado para a escrita do arquivo.

5) Inteiro que indica qual será o próximo registro lógico a ser acessado para leitura do arquivo.

6) Ponteiro para a tabela que contém todos os registros lógicos pertencentes ao arquivo.

7) Contador que indica o número de processos que estão com o arquivo aberto num dado instante. Um bit indica se o arquivo está aberto para a escrita (=1) ou para leitura (=0).

8) Contador que indica quantos registros estão pré-alocados para o arquivo.

9) Indica o número de registros lógicos utilizados pelo arquivo.

10) Data da criação do arquivo, válida para sistemas que possuam um relógio de tempo real.

4.3.1 - TABELA DE REGISTROS DO ARQUIVO

Como os arquivos são alocados continuamente no disco em segmentos compostos de vários registros cada, a tabela de registros lógicos contém o primeiro registro e o número de registros para cada segmento do arquivo. Como um segmento é composto por vários registros logicamente contíguos do disco, pode-se localizar todos os registros que fazem parte de cada arquivo.

Esta fica localizada junto ao descritor e tem um tamanho fixo definido na geração do sistema pelo parâmetro "nseg", através da primitiva "def-tcd", como será visto na Seção 4.6.1. A estrutura desta tabela é mostrada na Figura 4.2.

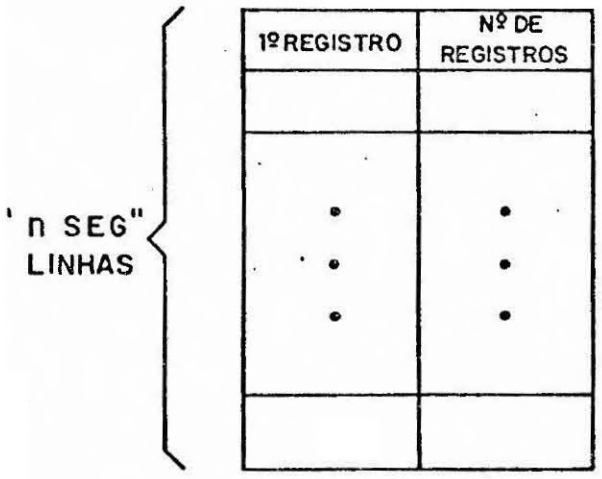


Fig. 4.2 - Estrutura dos registros do arquivo.

O número de segmentos de um arquivo é um parâmetro importante, pois vai determinar em quantas partes podem ser divididos os arquivos do sistema. Quanto mais particionados os arquivos, pior será a resposta do monitor aos pedidos de acesso a eles.

Idealmente, os arquivos deveriam ser formados por um único segmento, situação onde o acesso seria o mais otimizado possível, porém, o espaço do disco seria muito mal aproveitado (veja Seção 2.6). Existe, portanto, um compromisso entre o acesso rápido e o melhor aproveitamento do espaço do disco, que deve ser determinado para cada aplicação específica.

4.3.2 - LISTA DE CONTROLE DE ACESSO

Através desta lista é implementado o método de proteção de arquivos, citado na Seção 3.4.3. Na criação do arquivo, o processo fornece o identificador do processo e o tipo de acesso permitido para cada processo que irá acessar o arquivo. Estas informações são armazenadas junto ao descritor do novo arquivo, na forma de uma tabela a ser utilizada pelo S.B.A. para verificação da permissão do acesso.

O identificador do processo é um inteiro, de acordo com a convenção da máquina utilizada. O tipo de acesso é uma cadeia de até três caracteres "r", "w", "x" que indicam:

r = acesso apenas para leitura.

w = acesso apenas para escrita.

x = acesso apenas para execução.

Pode ser feita qualquer combinação dos caracteres r, w e x. Caso um arquivo seja público, ele ainda pode ser acessado seletivamente. Neste caso, a lista de permissão de acesso só é analisada em relação ao tipo de acesso. A lista de controle de acesso é mostrada na Figura 4.3.

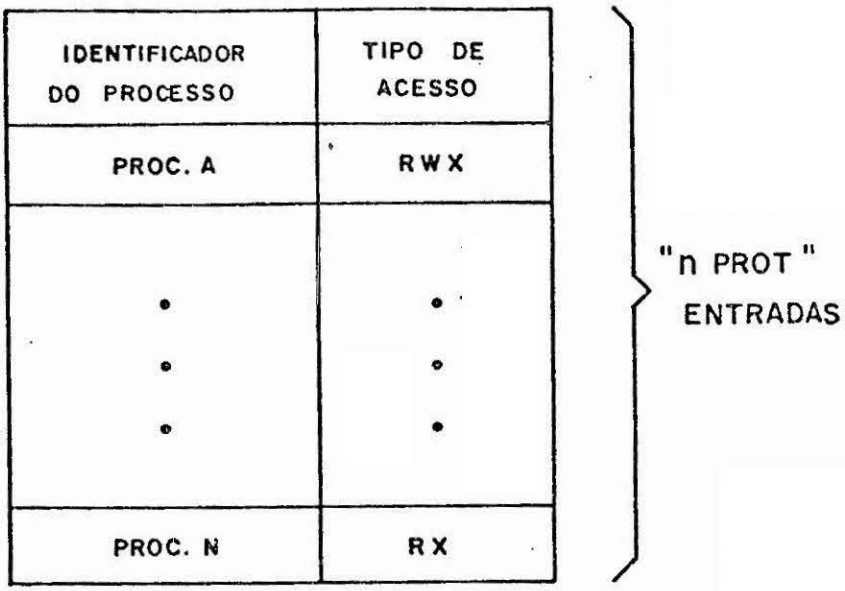


Fig. 4.3 - Lista de controle de acesso.

O tamanho da lista de controle de acesso é definido pelo parâmetro "nprot" através da primitiva "def-tcd". Este parâmetro é função do número máximo de arquivos que vão existir para cada aplicação e pode ser calculado como uma fração do número máximo de arquivos.

4.4 - DETALHAMENTO DO MÉTODO DE ALOCAÇÃO DE ESPAÇO

A alocação de espaço no disco é iniciada por um pedido de escrita a um registro lógico, feito ao S.B.A por um processo de aplicação através da primitiva "escr-reg".

Em primeiro lugar, essa primitiva verifica se o arquivo está sendo acessado para escrita por qualquer outro processo nesse instante. Em caso afirmativo ocorre um erro, o qual é devido a tentativa de acesso concorrente para a escrita.

Caso contrário, o vetor de bits do disco é consultado e mantido pelo S.O.A, que contém as informações sobre os blocos livres e ocupados a cada instante. O vetor de bits é armazenado no disco próximo ao diretório e aos descritores nas trilhas iniciais do disco.

Esse vetor é gerado na inicialização, durante a criação do diretório no disco, através da primitiva "cria-dir". Cada bit do vetor indica se um bloco do disco está livre (bit=0) ou ocupado (bit=1).

Embora o processo de aplicação requisite um registro lógico por acesso, o monitor aloca vários blocos (grupos de setores), pressupondo que estes irão ser utilizados logo em seguida. Esta técnica é conhecida como pré-alocação.

Desse modo, no descritor do arquivo existirá a indicação de que já estão alocados alguns registros adicionais para o arquivo e nos próximos pedidos de escrita não haverá necessidade de consulta ao vetor de bits, o que agiliza a alocação do disco melhorando o desempenho.

Além disso, esse método contribui para minimizar a fragmentação dos arquivos do monitor.

O algoritmo de alocação leva em conta também a posição atual do arquivo no disco, tentando alocar segmentos que sejam próximos

a outros já ocupados pelo arquivo. Com isto, pretende-se que os segmentos de um arquivo fiquem todos próximos uns dos outros, o que contribui para minimizar o tempo de acesso a este arquivo.

O tamanho do bloco de alocação a ser adotado e o número de blocos a serem pré-alocados são parâmetros definidos pelo usuário na fase de configuração, através da primitiva "def-tcd". O tamanho do bloco físico é um dos parâmetros de grande importância no desempenho do monitor. Ele é função de vários fatores como:

a) tipo do disco utilizado: cada tipo de disco possui um tamanho ótimo para ser acessado;

b) tipo do controlador: o "hardware" do controlador utilizado geralmente define uma unidade mínima de acesso ao periférico;

c) fatores de "software": são o tamanho do registro lógico desejado pela aplicação e o tamanho médio dos arquivos do sistema.

O número de blocos a serem pré-alocados ("nbpr") também é dependente do tamanho médio dos arquivos do sistema e, portanto, de cada tipo de aplicação. Deve-se observar que a pré-alocação só é efetiva quando é utilizada a alocação contínua do disco.

A pré-alocação é considerada como uma tarefa opcional, isto é, caso não existam blocos suficientes para ser pré-alocados num dado instante, isto não acarreta erro, pois apenas são pré-alocados os blocos disponíveis ou só o registro realmente pedido.

Os blocos pré-alocados e não utilizados são devolvidos ao sistema quando o arquivo é fechado. Isto influi no número de arquivos que podem estar abertos num mesmo instante, pois podem existir muitos blocos pré-alocados e não utilizados que diminuem virtualmente o espaço disponível para a abertura de novos arquivos.

Esses fatores devem ser examinados para a determinação do tamanho do bloco e do número de blocos a serem pré-alocados.

O algoritmo de alocação é implementado pelo S.O.A, o qual utiliza tabelas montadas na geração do sistema com as informações fornecidas pelos parâmetros passados para a primitiva def-tcd.

Podem existir várias tabelas, uma para cada disco utilizado, o que permite a acomodação de periféricos distintos simultaneamente. O conteúdo de cada tabela de controle do disco é mostrado na Figura 4.4.

Para melhor ilustrar o método de alocação adotado, a Figura 4.5 mostra o fluxograma simplificado ao algoritmo de alocação, salientando a pré-alocação. Este algoritmo é executado logo após ser chamada a primitiva "escr-reg".

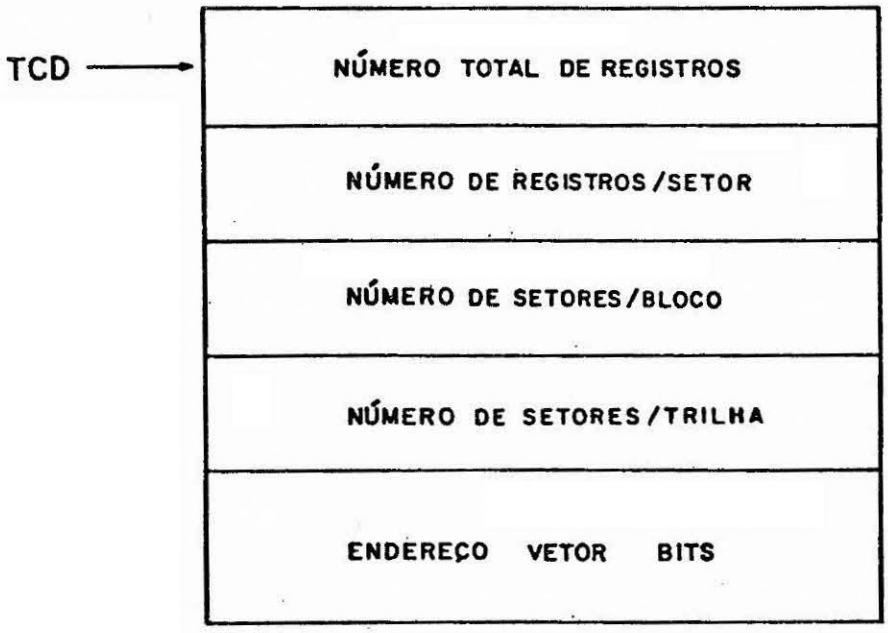


Fig. 4.4 - Estrutura de controle do disco.

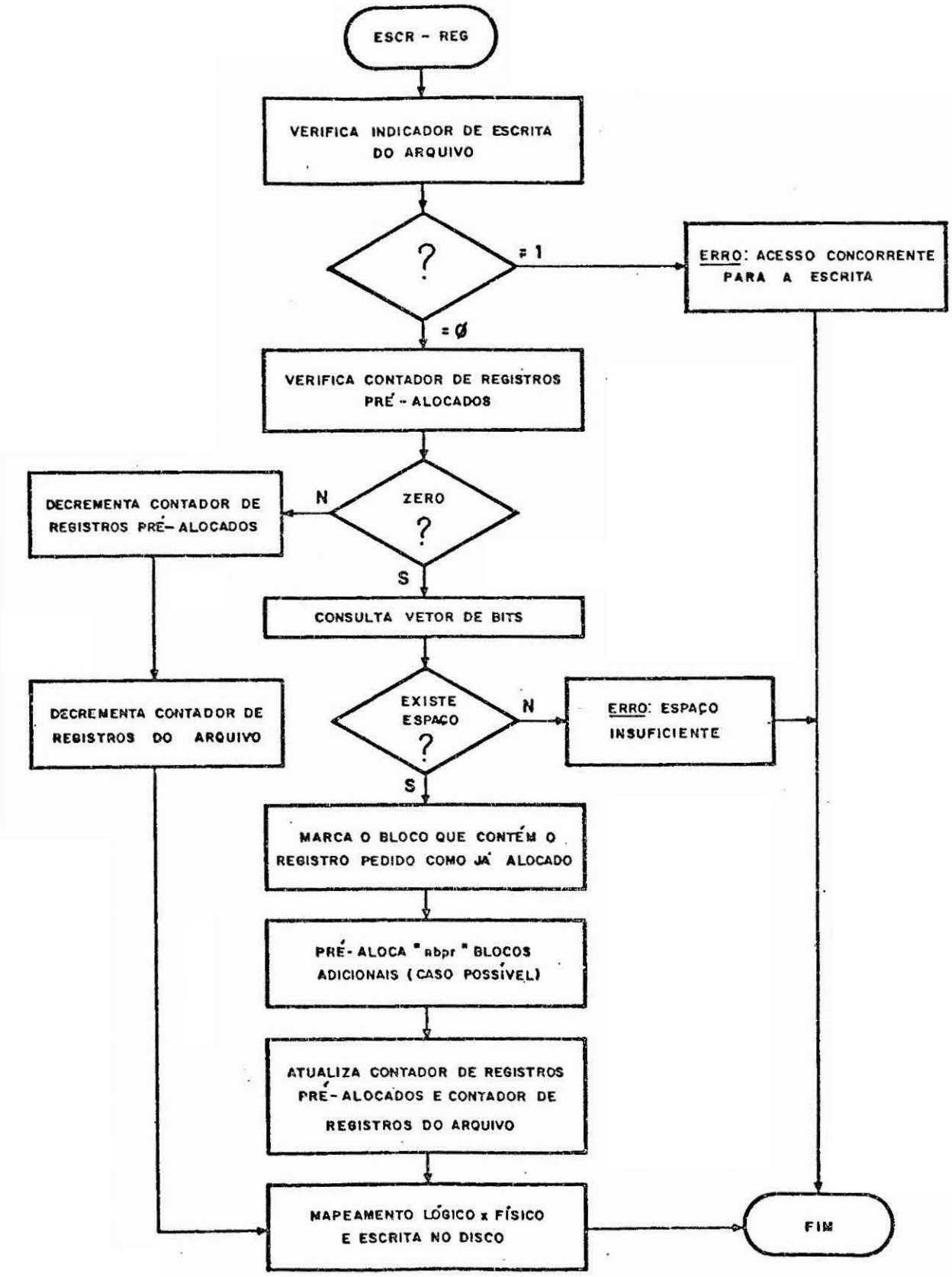


Fig. 4.5 - Fluxograma do algoritmo de alocação.

4.5 - DETALHAMENTO DAS ROTINAS DE OTIMIZAÇÃO

No nível 2 - C.E.S., são implementadas duas técnicas de otimização específicas:

1) otimização das transferências de dados do disco para a memória, utilizando um método de memória "cache";
2) escalonamento dos pedidos de acesso ao disco usando dois algoritmos: do tipo C-SCAN para operações de escrita e do tipo FCFS para operações de leitura (Seção 3.4.6). O escalonamento utiliza os "buffers" do "cache".

Essas técnicas são implementadas por um conjunto de rotinas baseadas no artigo de Mckeon (1985), adaptado para ser utilizado no Monitor de Disco.

A descrição detalhada do funcionamento das rotinas é feita a seguir nas Seções 4.5.1 e 4.5.2.

Essas rotinas de otimização são utilizadas nas primitivas "leit-set" e "escr-set" através da chamada a rotina "es-buf", que é uma rotina de E/S que utiliza o "cache".

Na realidade, as primitivas "escr-set" e "leit-set" consistem apenas em uma chamada a "es-buf", com um dos parâmetros indicando se a operação é de escrita ou de leitura.

A rotina "es-buf", invoca as outras rotinas auxiliares que implementam os algoritmos de transferência de dados ("cache") e a técnica de escalonamento.

Para acompanhar a descrição dos algoritmos, a Tabela 4.1 apresenta as rotinas de E/S com os respectivos parâmetros e funções. O pseudo-código destas rotinas é apresentado no Apêndice B.

TABELA 4.1

ROTINAS DE ENTRADA E SAÍDA DO MONITOR

NOME E PARÂMETROS	FUNÇÃO
es-buf (disp, ℓ/e, set trilh, end)	transfere um bloco de dados de um disco para os "buffers" ou vice-versa.
aloc-buf ()	aloca o "buffer" com a menor frequência de uso, que seja o menos usado recentemente (L.R.U.).
proc-buf (disp, set, trilh)	procura pelo "buffer" especificado na memória.
escr-disc ()	transfere para o disco todos os "buffers" de escrita pendentes, usando o escalonamento do tipo C-SCAN.
buf-escr ()	procura um "buffer" do tipo escrita na memória.
lib-buf ()	libera um "buffer", devolvendo-o para a lista de "buffers" livres.

Além das rotinas auxiliares, é necessário que o Sistema Operacional da máquina onde o monitor for implantado possua algumas rotinas utilitárias que farão a interface com o "hardware".

Essas rotinas utilitárias são de uso geral e normalmente disponíveis na maioria dos Sistemas Operacionais. A seguir, são descritas as três rotinas necessárias para a interface com o monitor:

1) *Rotina para transferência de dados entre dois endereços de memória.*

mov-dad (fonte, destino)

Esta rotina move uma área de dados do endereço "fonte" para o endereço "destino". O tamanho da área é igual ao tamanho do "buffer" usado pelo monitor, conforme definido na configuração do sistema.

2)*Rotina para entrada e saída do disco:*

esc-disp (disp, ℓ/e, set, trilh, end).

Esta rotina transfere um bloco de dados de/para um disco "disp" para/de uma área de memória iniciada no endereço "end". O sentido da transferência é determinado por "ℓ/e" (leitura/escrita). O bloco do disco inicia no setor definido por "(set, trilh)".

3) *Rotina de envio de mensagem de erro.*

msg-erro ("mensagem").

Esta rotina envia a mensagem entre aspas para o processo e interrompe sua execução. É passado um código de erro ao processo que deve tomar as providências necessárias.

4.5.1 - OTIMIZAÇÃO DAS TRANSFERÊNCIAS DE DADOS

O algoritmo de transferência de dados trabalha com uma determinada área de dados que é dividida em vários "buffers" com tamanho igual a um bloco físico. O tamanho desta área destinada ao "cache" e o tamanho dos "buffers" são parâmetros definidos na geração do sistema pela primitiva "def-ptd".

Os "buffers" são controlados pelo algoritmo por meio de um cabeçalho existente em cada um, o qual contém informações sobre o tipo de "buffer", o dono do "buffer" a frequência de acesso e o último acesso. Existe também a área de dados do tamanho de um bloco físico, como é mostrado na Figura 4.6. Os "buffers" podem ser de dois tipos:

a) *buffer de leitura:* que contém informações que foram lidas do disco ou já transferidas para o disco,

b) *buffer de escrita:* que contém informações que ainda não foram transferidas para o disco.

Na inicialização do monitor, os "buffers" são todos colocados em uma lista de "buffers" livres, organizada como uma lista conectada, através da primitiva def-ptd.

Todo acesso ao disco é feito através de uma chamada à rotina "*es-buf*" que vai verificar se a cópia do bloco desejado está num dos "buffers" da memória. Esta tarefa é realizada pela rotina "*proc-buf*".

Se a cópia do bloco é encontrada num "buffer" da memória ("hit"), incrementa-se o parâmetro de frequência de acesso deste "buffer" (XBFREQ) e faz-se a transferência de dados do "buffer" para a área de dados do processo que chamou a rotina.

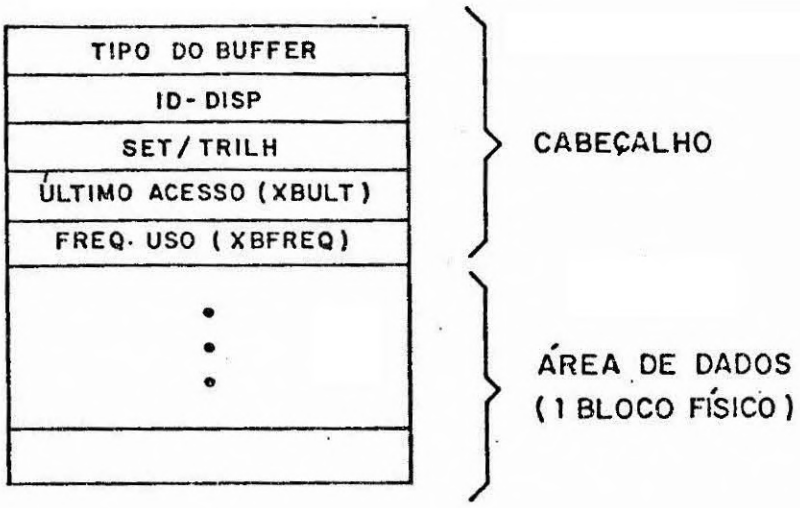


Fig. 4.6 - Estrutura dos "buffers".

Como as cópias de blocos que permanecerem na memória têm seus parâmetros de frequência de acesso (XBFREQ) sempre aumentados a cada chamada, é necessário que estes parâmetros sejam reduzidos periodicamente. Caso contrário, pode ocorrer que alguns blocos não possam mais ser removidos do "cache".

Essa redução de XBFREQ é implementada dividindo o parâmetro XBFREQ pela metade, após um determinado número de chamadas à rotina de alocação. O número de chamadas que pode ocorrer entre duas reduções é definido pelo parâmetro "nred" fornecido na hora da geração.

O parâmetro "nred" depende da aplicação e influi na velocidade de resposta do algoritmo, juntamente com o tamanho do bloco e da área total do "cache". Se a área total é aumentada, o algoritmo deve responder mais lentamente às mudanças da demanda, e o fator de redução deve ser diminuído.

Um ponto importante é verificar qual é o valor máximo do parâmetro de frequência de uso (XBFREQ), senão pode ocorrer um "overflow" caso um bloco seja muito acessado pelos processos. Para o exemplo citado em Mckeon (1985), como o parâmetro XBFREQ é aumentado em incrementos de 128 unidades a cada acesso e reduzido pela metade a cada 16 acessos, seu valor máximo neste caso vai tender a 2x16x128 = 4096, que é um valor seguro.

Se a cópia do bloco desejado não foi encontrada no "cache" ("miss"), a rotina de alocação de "buffer" *aloc-buf* é chamada. Esta rotina consulta em primeiro lugar a lista de "buffers" livres. Caso não exista nenhum livre, a rotina procura um "buffer" entre os ocupados para ser substituído. Procura-se o "buffer" de leitura que possua a menor frequência de uso (XBFREQ) para ser substituído.

Se existirem vários "buffers" com o mesmo XBFREQ, então, procura-se o que não tenha sido acessado a mais tempo, o que é indicado pelo parâmetro de último acesso (XBULT).

Se não existirem "buffers" de leitura no "cache", ou se o número de "buffers" de escrita é maior que o permitido pelo parâmetro "nbe" definido na configuração, então, deve ser feita a escrita no disco.

Esta tarefa é realizada pela rotina de escrita com escalonamento *escr-disc* (veja Seção 4.5.2). Assim o algoritmo volta ao seu início para nova tentativa de encontrar um "buffer" a ser substituído.

O "buffer" encontrado é colocado na lista de "buffers" livres pela rotina *lib-buf* e será substituído por um novo "buffer" a ser lido do disco.

No algoritmo de Mckeon (1985), lê-se apenas um setor do disco a cada acesso. No monitor, faz-se antecipadamente a leitura de vários setores do disco, ou seja, de um bloco físico, modificando-se o algoritmo original. Esta modificação aumenta o desempenho do algoritmo, pois aproveita o tempo gasto no movimento de busca da trilha/setor desejado para fazer a leitura de vários setores de uma só vez.

Supõe-se que seja utilizada a alocação contínua do disco e exista pouca fragmentação dos arquivos para que a leitura antecipada tenha o efeito desejado.

4.5.2 - ESCALONAMENTO DO DISCO

O algoritmo de escalonamento para as operações de escrita funciona juntamente com o método "cache". Os dados que serão escritos no disco são agrupados em "buffers" do tipo escrita, os quais são acumulados para ser transferidos de uma só vez para o disco. Esta tarefa é feita pela rotina de escrita no disco *escr-disc*, chamada pela rotina aloc-buf.

O número de "buffers" de escrita que pode existir antes desta ser iniciada é determinado pelo parâmetro "nbe", definido pela primitiva "def-ptd". Este parâmetro tem grande influência no desempenho do algoritmo de escalonamento, pois determina a frequência com que são feitas as atualizações do disco.

Se esse número é muito pequeno, são feitas mais escritas e o desempenho diminui. Se ele for muito grande, o sistema acumula muitas informações nos "buffers" antes de passá-las para o disco, o que causa os seguintes problemas:

- o sistema fica mais vulnerável a erros de escrita que possam ocorrer nas transferências físicas,.

- se ocorrer um problema de "software" ou "hardware" na máquina, é mais provável que o disco não tenha sido atualizado com as informações dos "buffers", embora para os processos esta operação já tenha sido executada.

Para determinar o "buffer" a ser escrito no disco é chamada a rotina *buf-escr*. Esta rotina retorna com o endereço do "buffer" que possui o menor identificador entre os que vão ser escritos, ou seja, o endereço do "buffer" que seja associado ao menor setor da trilha mais externa (por exemplo: setor 1 da trilha 0).

Desse modo as escritas no disco são feitas em um só sentido do braço do acionador, o que minimiza os movimentos necessários.

Essa técnica implementa um algoritmo do tipo C-SCAN, pois as escritas são feitas em um único sentido, partindo da trilha mais externa para a mais interna do disco (veja Seção 2.8.3).

O escalonamento nas operações de leitura utiliza o método do FCFS ("first-come-first-served") devido à natureza aleatória dos pedidos de leitura, método este que torna ineficaz qualquer outro mais elaborado.

As operações de leitura são executadas pela rotina *es-buf*, e a otimização da leitura é obtida pela leitura antecipada de setores do disco, como descrito na Seção 2.7.5.

Para compreender melhor as relações entre as várias rotinas descritas, a Fig. 4.7 mostra a interação entre elas, as primitivas do monitor e as rotinas do Sistema Operacional.

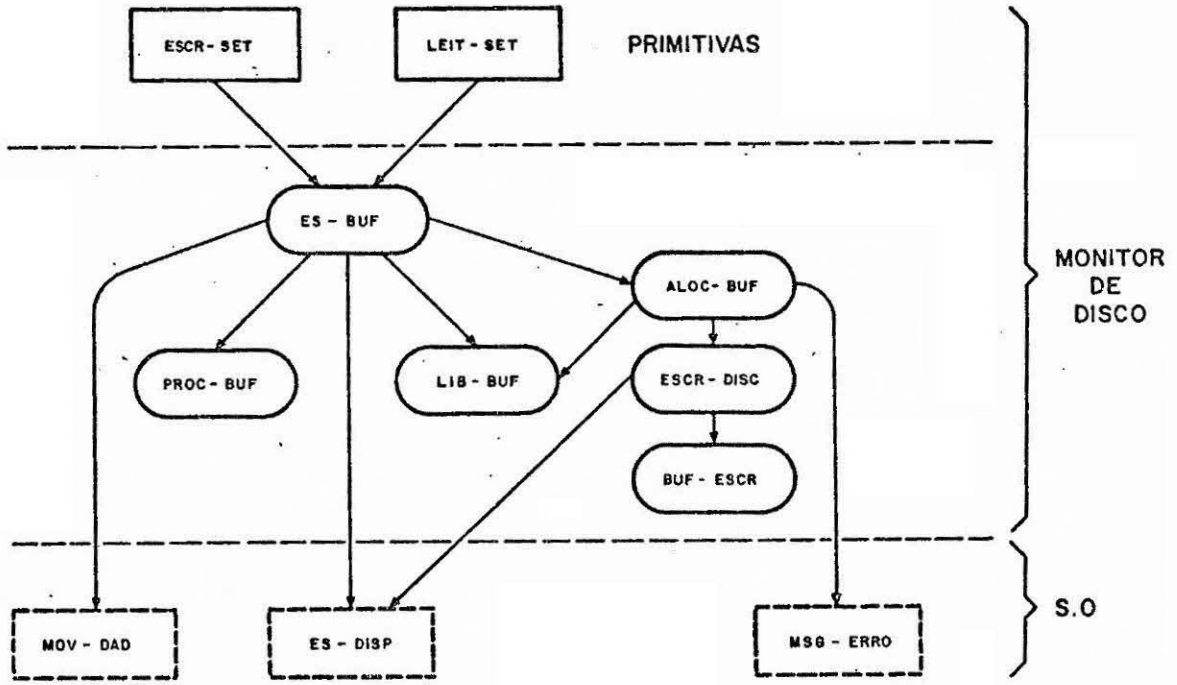


Fig. 4.7 - Rotinas de otimização.

4.6 - DETALHAMENTO DAS PRIMITIVAS

Nesta seção são detalhadas as primitivas do Monitor de Disco apresentadas no Capítulo 3. Estas primitivas são divididas em dois grandes grupos: primitivas de configuração e primitivas de manipulação de arquivos.

As primitivas de configuração são utilizadas na fase de inicialização do monitor, para o estabelecimento dos seus vários parâmetros de operação por parte do usuário.

As primitivas de manipulação são utilizadas na fase de execução, o que permite o acesso aos arquivos do monitor por parte dos processos de aplicação.

4.6.1 - PRIMITIVAS DE CONFIGURAÇÃO

As primitivas de configuração do monitor são:

1) define as tabelas de controle do disco,

2) define os parâmetros de transferência de dados,

3) cria o diretório,

4) inicializa o Monitor de Disco.

Estas primitivas são executadas utilizando as rotinas já disponíveis no Sistema Operacional da máquina, relacionadas na Seção 4.5.

1) *Define as tabelas de controle do disco:*

def-tcd (disp, ntrl, tam-reg, tam-set, tam-bl),

disp = identificação do disco;

ntrl = número total de registros lógicos do disco;

tam-reg = tamanho do registro lógico, em "bytes";

tam-set = tamanho do setor físico, em registros lógicos;

tam-bl = tamanho do bloco físico, em setores físicos.

Esta primitiva é chamada para definir os vários parâmetros de blocagem (Tabela 3.1) do monitor. O parâmetro "ntrl" multiplicado por "tam-reg" exprime a capacidade do disco em "bytes".

Estes parâmetros são utilizados para a criação de tabelas do Sistema de Organização de Arquivos - S.O.A., o qual faz o mapeamento lógico versus físico do disco. A definição dos parâmetros de blocagem deve ser feita pelo usuário de modo a melhor se adaptar a fatores como:

a) tamanho do registro lógico ótimo para uso na aplicação desejada: cada aplicação tem uma necessidade diferente em relação ao tamanho do registro a ser acessado, de acordo com o tipo e velocidade da transferência dos dados;

b) tamanho do setor físico ótimo para o controlador e periférico utilizados: de acordo com as características físicas e a capacidade do periférico existe um tamanho do setor que otimiza os acessos ao mesmo;

c) tamanho do bloco físico: esse parâmetro é função do tamanho dos "buffers" adotados para o método "cache" e para o método de alocação do disco.

Esta primitiva deve ser a primeira a ser executada pelo usuário na configuração do sistema.

2) *Define os parâmetros de transferência de dados:*

def-ptd (nbuf, tam-buf, nred, nbe).

nbuf = número máximo de "buffers";

tam-buf = tamanho dos "buffers", em blocos lógicos;

nred = número de acessos sem redução da frequência de acesso;

nbe = número máximo de "buffers" de escrita permitidos, antes da atualização do disco.

Esta primitiva define os parâmetros a serem utilizados pelo nível 2 - Controle de E/S. Este nível implementa o método de transferência de dados e o escalonamento do monitor. Sua principal função é alocar um espaço de memória suficiente e criar uma lista de "buffers" para o sistema "cache". Em seguida os "buffers" são inicializados, isto é, seus cabeçalhos são preenchidos com os parâmetros adequados.

O parâmetro "nbuf" define o número de "buffers" que serão criados na inicialização. Este parâmetro depende da área de memória disponível para o "cache" e varia em cada configuração.

O parâmetro "tam-buf" é expresso em blocos físicos, isto é, cada "buffer" possui (tam-buf x tam-bl) "bytes".

O número de acessos ocorridos antes da redução da frequência de acesso é definido pelo parâmetro "nred". Este parâmetro determina a rapidez da resposta do "cache" aos pedidos de acesso.

Por exemplo, caso o tamanho da memória disponível para o "cache" seja aumentado, este pode responder mais lentamente, o que é feito aumentando o número de acessos entre as reduções "nred".

O parâmetro "nbe" define o número máximo de "buffers" do tipo escrita que podem existir na memória antes de ser transferidos para o disco. Este parâmetro influencia diretamente o algoritmo utilizado para fazer a escrita no disco, algoritmo este que realiza o escalonamento dos movimentos do braço do acionador do disco.

Esta primitiva deve ser executada após a primitiva "def-tcd" e antes da "cria-dir" durante a configuração.

3) *Cria o diretório:*

cria-dir (disp, narq, nativ, nbpr, nseg, nprot).

disp = disco onde vai ser criado o diretório;

narq = número total de arquivos do sistema;

nativ = número máximo de arquivos ativos (abertos) simultaneamente;

nbpr = número de blocos a serem pré-alocados;

nseg = número de segmentos de um arquivo;

nprot = tamanho da lista de controle de acesso.

Esta primitiva é chamada para a criação de um arquivo diretório no disco "disp". O arquivo diretório possuirá "narq" entradas e o monitor poderá suportar até "nativ" arquivos abertos simultaneamente. Para tanto, é reservada uma área de memória suficiente para acomodar os descritores dos "nativ" arquivos que poderão estar abertos ao mesmo tempo.

Cria-se o vetor de bits associados ao disco. Este vetor é inicializado, o que indica que todo o disco está livre.

O parâmetro "nbpr" é utilizado pelo algoritmo de pré-alocação de blocos durante uma escrita no disco. Ele indica qual a quantidade de blocos lógicos deve ser pré-alocada pelo monitor.

O parâmetro "nseg" vai definir o número máximo de segmentos que pode possuir cada arquivo, o que define o tamanho da área reservada no descritor para o armazenamento da tabela de registros do arquivo, esta descrita na Seção 4.3.1.

O parâmetro "nprot" define o tamanho da lista de proteção de acesso e indica o número de processos que podem acessar os arquivos do sistema (veja Seção 4.3.2).

Esta primitiva deve ser executada após "def-tcd" e "def-ptd" na fase de configuração.

4) *Inicializar o Monitor de Disco:*

inic-MD ()

Esta primitiva finaliza a fase de configuração e inicializa todas as variáveis, entrando na fase de èxecução do monitor, em que todas as outras primitivas de configuração já devem ter sido executadas.

4.6.2 - PRIMITIVAS DE MANIPULAÇÃO

As várias primitivas de manipulação citadas no Capítulo 3 são descritas a seguir:

1) *Cria arquivo:*

cria-arq (nome-arq, disp, contr-acesso, id-proc)

nome-arq = nome do arquivo a ser criado;

disp = identificação do disco a ser acessado;

contr-acesso = informações de proteção de acesso para o novo arquivo;

id-proc = identificação do processo criador do arquivo.

Esta primitiva é utilizada por um processo para a criação de um novo arquivo. O nome do arquivo não deve ainda existir no diretório do disco "disp" especificado. O parâmetro "id-proc" indica qual processo está criando o arquivo.

O parâmetro "contr-acesso" é uma lista com o seguinte formato:

id-proc 1: rwx, id-proc 2: rwx,..., id-proc n: rwx

Nesta lista estão especificados os processos e os tipos de acesso permitidos ao arquivo que está sendo criado. Os identificadores de processos são inteiros que especificam os processos de 1 a "n". O tipo de acesso é formado por, no máximo, três caracteres que indicam:

r - acesso apenas para leitura;

w - acesso apenas para escrita;

x - acesso penas para execução.

Um arquivo que deve ser acessado apenas para execução possui apenas um "x" para todos os processos. Estas informações são armazenadas em uma lista localizada no descritor do arquivo. Se um arquivo pode ser acessado por todos os processos, ele possui a palavra "público " no nome dos processos, seguido pelo tipo de acesso r, w ou x.

2) *Remove arquivo:*

rem-arq (nome-arq, disp, id-proc)

nome-arq = nome do arquivo a ser removido;

disp = identificação do disco a ser acessado;

id-proc = identificação do processo.

Esta primitiva remove o nome do arquivo "nome-arq" do diretório do disco "disp". O conteúdo do arquivo não pode mais ser referenciado, pois não possui entrada no diretório.

Todo arquivo pode estar sendo acessado, ao mesmo tempo, por vários processos, o que é indicado por um contador de processos localizado no descritor do arquivo. Toda chamada à primitiva de remoção decrementa este contador. Um arquivo só pode ser removido caso seu contador esteja com valor zero.

3) *Abre arquivo:*

abre-arq (nome-arq, disp, tipo-acesso, id-proc).

nome-arq = nome do arquivo a ser aberto;

disp = identificação do disco;

tipo-acesso = letra r, w ou x que indica o tipo de acesso desejado para o arquivo;

id-proc = identificação do processo.

Esta primitiva prepara o arquivo específico, localizado no disco "disp", para operações de E/S determinadas por "tipo-acesso".

A abertura de um arquivo consiste em localizar o arquivo e, caso exista, buscar seu descritor e verificar a validade do acesso desejado. Caso não existam problemas, o descritor do arquivo é colocado na lista de descritores ativos, o qual fica pronto para acessos de leitura ou escrita.

4) *Fecha arquivo*

feche-arq (nome-arq, disp, id-proc)

nome-arq = nome do arquivo a ser fechado;

disp = identificação do disco;

id-proc = identificação do processo.

Esta primitiva prepara o arquivo para ser desativado, isto é, retira seu descritor da lista de arquivos ativos. Se o arquivo estiver sendo acessado por outros processos, ele não pode ser fechado. Neste caso, apenas o contador de processos deste arquivo é decrementado.

Se o arquivo for fechado, todos os "buffers" com dados do arquivo devem ser escritos no disco e o diretório também deve ser atualizado com as informações sobre o tamanho do arquivo.

5) *Monta descritor do arquivo:*

monta-descr (id-arq, disp, contr-acesso, id-proc).

id-arq = identificador do arquivo;

disp = identificação do disco;

contr-acesso = informações de proteção de acesso;

id-proc = identificação do processo.

Esta primitiva é de uso interno do monitor, utilizada para interfacear o S.L.A com o S.B.A. Sua função é montar um descritor para o arquivo com o identificador "id-arq" e preenchê-lo com as informações enviadas pelo S.L.A.

A primitiva "cria-arq" faz uma chamada à primitiva "monta-descr" para a criação do descritor propriamente dito.

6) *Altera descritor do arquivo:*

alt-descr (id-arq, disp, desloc, oper, novo-val).

id-arq = identificador do arquivo;

disp = identificação do disco;

desloc = deslocamento que indica a posição, dentro do descritor, do parâmetro a ser alterado;

oper = operação a ser realizada, que pode ser:

incr: incrementa o parâmetro,

descr: decrementa o parâmetro,

subs : substitui o parâmetro.

novo-val = novo valor do parâmetro no caso de substituição.

Esta primitiva é de uso interno do monitor, utilizada para interfacear o S.L.A. e o S.B.A. Sua função é fazer a alteração do parâmetro identificado por "desloc" no descritor do arquivo com o "id-arq" especificado. A operação a ser efetuada é especificada em "oper" e, caso seja uma substituição, o parâmetro antigo é substituído por "novo-val"; caso contrário, este parâmetro é ignorado.

7) *Leitura de registro lógico:*

leit-reg (id-arq, disp, nreg, id-proc).

id-arq = identificador do arquivo a ser lido;

disp = identificação do disco;

nreg = número do registro lógico a ser acessado;

id-proc = identificador do processo.

Esta primitiva, quando invocada por um processo de aplicação "id-proc", faz a leitura do registro lógico "nreg" que pertence ao arquivo "id-arq" armazenado no disco "disp". As informações são armazenadas na área de dados pertencente ao processo chamador. O mapeamento do número do registro lógico para o setor físico a ser lido é feito pelo nível 3 - S.O.A.

8) *Escrita de registro lógico:*

escr-reg (id-arq, disp, nreg, id-proc).

id-arq = identificador do arquivo;

disp = identificador do disco;

nreg = número do registro lógico a ser acessado;

id-proc = identificador do processo.

Esta primitiva, ao ser invocada pelo processo "id-proc", escreve no registro lógico "nreg" do arquivo "id-arq" que está armazenado no periférico "disp". As informações são transferidas ao disco a partir da área de dados dedicada ao processo chamador. O mapeamento do registro lógico para o setor físico a ser escrito é feito pelo monitor através do nível 3 - S.O.A.

Além disso, esta primitiva é responsável pela alocação do espaço no disco. Quando possível, a pré-alocação do espaço para o arquivo, é implantada com base no parâmetro "nbpr" definido na configuração (veja primitiva "def-tdc").

9) *Leitura do setor físico:*

leit-set (disp, set, trilh, end).

disp = identificação do disco;

set = número do setor a ser acessado;

trilh = número da trilha a ser acessada;

end = endereço inicial da área de transferência dos dados.

Esta primitiva faz a leitura do setor físico "set", localizado na trilha "trilh" do periférico "disp", transferindo os dados deste setor para a área de memória com endereço inicial "end". É uma primitiva interna do monitor, chamada por todas as primitivas que desejam ler o disco.

10) *Escrita do setor físico:*

escr-set (disp, set, trilh, end).

disp = identificação do disco;

set = número do setor a ser acessado;

trilh = número da trilha a ser acessada;

end = endereço inicial da área para a transferência dos dados.

Esta primitiva transfere os dados da área de memória iniciada no endereço "end" para o setor "set" da trilha "trilh" do periférico "disp". É uma primitiva interna do monitor, chamada por outras primitivas que desejam escrever no disco.

4.7 - MODO DE IMPLEMENTAÇÃO DO MONITOR DE DISCO

O monitor foi concebido para ser implantado em uma máquina onde existem vários processos que operam concorrentemente, competindo com os demais processo pelo uso dos discos magnéticos.

Todos os pedidos de acesso aos discos devem ser passados ao monitor que, através da interface com o núcleo do sistema operacional, vai garantir o uso eficiente dos discos magnéticos.

Considera-se, então, que o núcleo possui mecanismos de comunicação e de sincronização que vão ser utilizados para interfacear o monitor com a máquina.

Além disso, devem existir rotinas de E/S ("drivers") que são responsáveis pela interface entre os controladores dos discos magnéticos e o nível 2 (C.E.S.) do Monitor de Disco.

O Monitor de Disco pode ser implementado de duas maneiras:

1) *Monitor passivo:* como no conceito de Hoare (1974), é uma entidade passiva, ativada somente quando chamada por um processo de aplicação.

2) *Monitor ativo:* implementado como um processo que compete com os demais processos pelo uso do processador e é escalonado pelo núcleo do sistema operacional.

Em ambas as alternativas acima, o Monitor de Disco não deve ser implementado como pertencendo ao núcleo, devido às seguintes razões:

- o núcleo deve ser simples e compacto, o que não ocorreria caso os problemas de tratamento dos discos fossem incorporados a ele;

- o monitor deve possuir o maior grau de portabilidade possível, podendo ser implantado em várias máquinas. Este fato leva a uma implementação que não seja diretamente ligada ao núcleo, mas a um nível acima deste.

A seguir, são analisados os modos de implementação citados. Para a análise, considera-se um núcleo que faça o escalonamento preemptivo dos processos, utilizando uma base de tempo que interrompe o processador a intervalos regulares. Esta interrupção não deve ser desabilitada.

4.7.1 - IMPLEMENTAÇÃO COMO UM MONITOR PASSIVO

Neste caso, todos os processos entram no monitor através da chamada a uma de suas primitivas com os parâmetros adequados.

A execução das primitivas está sujeita ao escalonamento da CPU, pois a prioridade do monitor é a mesma do processo que tem a sua posse.

O problema desta alternativa é que, caso o monitor esteja sendo executado por um processo de baixa prioridade, outros processos de mais alta prioridade não podem ter acesso a ele, mesmo que ganhem a posse do processador.

Portanto, esta solução não é adequada a aplicações de tempo real, onde existam processos de alta prioridade que precisam fazer acessos rápidos ao disco.

4.7.2 - IMPLEMENTAÇÃO COMO UM PROCESSO ATIVO

Nesta alternativa, quando um processo de aplicação deseja acessar o disco, ele deve fazê-lo utilizando um mecanismo de troca de mensagens existente no sistema operacional, para o envio de uma mensagem que contenha o tipo da operação desejada e os demais parâmetros.

Neste caso, o processo monitor possuirá uma prioridade fixa, determinada na inicialização do sistema. Esta prioridade é atribuída ao Monitor de Disco de acordo com sua importância para a aplicação.

O processo monitor pode ser interrompido pelo escalonador do núcleo como os demais processos de aplicação. Entretanto, se a sua prioridade for alta, ele sempre terá a posse do processador. Isto assegura um tempo de atendimento menor do que no caso da implementação como um monitor passivo.

Quando um processo de aplicação envia uma mensagem ao monitor, seu pedido é colocado na fila de espera associada ao monitor que vai servir um após o outro. Este mecanismo assegura a execução das primitivas em regime de exclusão mútua.

4.8 - INTERFACE COM O NÚCLEO

Pelo que foi exposto na seção anterior, conclui-se que o Monitor de Disco deve ser implementado como um processo, cuja prioridade pode ser definida de acordo com a sua aplicação.

Para que seja feito o interfaceamento do monitor com o sistema operacional da máquina, este deve fornecer os seguintes recursos:

1) *Mecanismo de troca de mensagens:* permite a comunicação entre o Monitor de Disco e os processos de aplicação.

2) *Rotina de E/S ("drivers"):* o tratamento dos discos propriamente dito é feito pelo sistema operacional. Esta rotina foi citada na Seção 4.5.

3) *Rotinas de tratamento de interrupção* associadas aos discos magnéticos. Elas são necessárias caso os controladores dos discos utilizem interrupção.

4) *Mecanismos de sincronização:* fazem a sincronização das rotinas de interrupção com as rotinas de E/S do sistema operacional.

5) *Rotinas utilitárias:* além das rotinas citadas na Seção 4.5, uma rotina de reserva de áreas de memória a ser utilizada na fase de configuração é necessária para o armazenamento dos diretórios, tabelas e "buffers" na memória.

CAPÍTULO 5

CONCLUSÃO

Apesar da teoria de sistemas de arquivos ser bastante abordada na literatura sobre sistemas operacionais, não existem trabalhos que tratem especificamente, com detalhes de implementação, do caso de sistemas de arquivos voltados para aplicações em tempo real, onde é essencial obter um alto desempenho no tratamento de arquivos.

Este trabalho tenta preencher esta lacuna de duas formas:

- apresentando um resumo dos conceitos envolvidos no projeto e na implementação de sistemas de arquivos, com ênfase nos aspectos importantes para a obtenção de um alto desempenho;

- elaborando uma proposta para a implementação de um monitor de disco de Alto Desempenho para aplicações em sistemas de tempo real, utilizando os conceitos apresentados.

Além da simplificação das estruturas de dados necessários, o Monitor de Disco utiliza uma técnica de memória "cache" para a minimização do número de acessos ao disco. Realiza também o escalonamento dos pedidos de acesso ao disco para a minimização dos movimentos da cabeça dos acionadores de discos.

Outra característica importante do monitor aqui proposto é a sua capacidade de reconfiguração, a qual permite que seus principais parâmetros de operação sejam programados para melhorar seu desempenho nas diferentes aplicações.

As soluções adotadas neste trabalho são variações de técnicas encontradas em outros sistemas existentes.

Procurou-se conceber o monitor para que ele pudesse ser facilmente transportado para diferentes máquinas, através de uma interface bem definida com o sistema operacional hospedeiro. Entretanto, sua portabilidade é limitada, pois ela depende das várias rotinas do sistema operacional necessárias para o interfaceamento. Estas rotinas são citadas no texto e nos apêndices deste trabalho.

Como sugestão para o prosseguimento do trabalho, propõe se que seja realizada a programação das primitivas em uma linguagem de alto nível e sua posterior implantação em um sistema de tempo real existente.

REFERÊNCIAS BIBLIOGRÁFICAS

CHANEY, R.; JOHNSON, B. Maximizing hard-disk performance. *Byte*, 9(5): 307-334. May 1984.

CHAPIN, N. A comparison of file organization techniques. In: ACM NATIONAL CONFERENCE, 24., New York, 1969. *Proceedings*, New York, N.Y., A.C.M., 1969. p.273-283.

DENNING, P.J. Effects of scheduling on file memory operations. In: AFIPS, 1967 SPRING JOINT COMPUTER CONFERENCE, *Proceedings*. AFIPS, 1967. v.30, p.9-21.

DIJKSTRA, E.W. The estructure of the "THE" multiprogramming system. *Communications of the ACM*, 3(5): 341-346, May 1968.

DODD, G.G. Elements of data management systems. *Computing Surveys*, 1(2): 117-133, June 1969.

FRANK, H. Analisys and optimization of disk storage devices for time-sharing systems. *Journal of the ACM*, 16(4): 602-620, Oct.1969.

FREEMAN, D.E.; PERRY, O.R. *I/O design:* data management in operating systems. New Jersey, N.J., Hayden Book, 1977.

GOTLIEB, C.C.; MAC EWEN, G.H. Performance of movable head disk storage devices. *Journal of the ACM*, 20(4): 604-623, Oct. 1973.

GUARINO REID, L.; KARLTON, P.L. A file system supporting cooperation between programs. In: ACM SYMPOSIUM ON OPERATING SISTEMS PRINCIPLES, 9., Bretton Woods, N.H., Oct. 1983. *Proceedings*, p.20-29.

HANSON, O. *Design of computer data files*. London, Computer Science, 1982.

HOGAN, T. Aspectos técnicos do CP/M. In: *Osborne CP/M Guia do Usuário*. São Paulo, Mc Graw-Hill, 1983.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS (IEEE). Data transfer. In: *Standard specification for microprocessor operating systems interface*. New York, N.Y. June 1984. p.23-41. (P-855/Draft 6.2).

KNUTH, D.E. *The art of computer programming: fundamental algorithms.* Reading, MA, Addison-Wesley, 1973.

LYNCH, W.C. Do disk arms move?. *Performance Evaluation Review*, 1(4): 3-16, Dec. 1972.

MADNICK, S.E.; ASLOP, J.W. A modular approach to file system design. In: AFIPS 1969 SPRING JOINT COMPUTER CONFERENCE, Boston, MA, May 14-16, 1969. *Proceedings*. Montvale, NJ, AFIPS, 1969 v.34, p.1-13.

McKEON, B. An algorithm for disk caching with limited memory. *Byte*, 10(9): 129-138, Sept. 1985.

PECHURA, M.A.; SCHOEFFLER, J.D. Estimating file access time of floppy disk. *Communications of the ACM*, 26(10): 754-763, Oct. 1983.

PETERSON, J.L. SILBERSCHATZ, A. *Operating systems concepts*. Reading, MA, Addison-Wesley, 1983.

POWELL, M. The DEMOS file system. In: ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, Nov. 1977. *Proceedings*. New York, NY, 1977. p.33-42.

RITCHIE, D.M.; THOMPSON, K. The UNIX time-sharing system. *Communications of the ACM*, 17(7): 365-375, July 1974.

SHAW, A.C. *The logical design of operating systems*. New Jersey, N.J. Prentice-Hall, 1974.

SHORE, J.E. On the external storage fragmentation produced by first-fit and best-fit allocation strategies. *Communications of the ACM*, 18(8): 433-440, Aug. 1975.

SMITH, A.J. Input/output optimization and disk architectures: a survey. *Performance Evaluation*, 1(2): 104-117, 1981.

TEOREY, T.J.; PINKERTON, T.B. A comparative analysis of disk scheduling policies. *Communications of the ACM*, 15(3): 177-184, Mar. 1972.

TEOREY, T.J. Properties of disk scheduling policies in multiprogrammed computer systems. In: AFIPS 1972 FALL JOINT COMPUTER CONFERENCE, *Proceedings*. Montvale, NJ, AFIPS, 1972. v.41 part 1, p.1-11.

BIBLIOGRAFIA

ABATE, J.; DUBNER, H. Optimizing the performance of a drum like storage. *IEEE Transactions on Computers*, C-18(11): 992-997, nov. 1969.

BOWEN, B.A.; BUHR, R.J.A. *The logical design of multiple-microprocessor systems*. Englewwod Cliffs, N.J., Prentice-Hall, 1980.

BRINCH-HANSEN, P. A case study: RC 4000. In: *Operating systems principles*. New Jersey, N.J. Prentice-Hall, 1973, p.237-285.

BRINCH-HANSEN, P. A programming methodology for operating system design. In: IFIP CONGRESS 74, Stockholm, Sweden, 1974. *Proceedings*. Amsterdam, Information Processing, 1974. p.394-397.

CRUCE, A.C.; ALEXANDER, S.C. Building a hard-disk interface for S-100; *Byte* 8(3): 130-148, Mar., (4): 304-321, Apr. (5): 368:382, May, 1983.

DALEY, R.C.; NEUMANN, P.G. A general-purpose file system for secondary storage. In: FALL JOINT COMPUTER CONFERENCE, Las Vegas, NV, 1965. *Proceedings*. New York, AFIPS Press, 1967. p.213-229

GROSSMAN, C.P.G. Cache-DASD storage design for improving system performance. *IBM System Journal*, 24(3-4): 316-334, 1985.

HANSON, D.R. A portable file directory system. *Software Practice and Experience*, 10(8): 623-634, 1980.

HARKER, J.M.; BREDE, D.W.; PATTISON, R.E.; SANTANA, G.R.; TAFT, L.G. A quarter century of disk file innovation. *IBM Journal of Research and Development*, 25(5): 677-689, Sept. 1985.

HOARE, C.A.R. Monitors: an operating systems structuring concept. *Communications of the ACM*, 17(10): 549-557, Oct. 1974.

HOFRI, M. Disk scheduling: FCFS x SSTF revisited. *Communications of the ACM*, 23(11): 645-653, Nov. 1980.

JACKSON, D.L.; COWAN, J. The proposed IEEE 855 microprocessor operating systems interface standard. *IEEE micro*, 4(4): 63-71 Aug. 1984.

KOP, H. Very smart hard disk controller offloadas host microcomputer. *Electronics*, 53(25): 140-145, Nov.20, 1980.

MCKENDRY, M.S. The use of monitors in microprocessor software development. *Euromicro Journal*, 4(5): 257-264, 1978.

PIRKOLA, G. A file system for a general-purpose time-sharing environment. *Proceedings of IEEE*, 63(6): 918-924, June 1975.

STOCK, T. Floppy disc file management. *Microprocessors and Microsystems*, 3(2): 113-115, Mar. 1979.

THOMPSON, K. UNIX implementation. *Bell Systems Technical Journal*, 57(6): 1931-1946, July/Aug. 1978.

THURBER, K.J.; DEWARD, R.C.; PETSCHAUER, T.W.; FEDDE, M.P.; KAMINSKI, D.G. I/O Concepts for a real time system. In: IEEE COMPUTER CONFERENCE, Washington, D.C., Fall, 1976. *Proceedings*. s.l., IEEE Computer Society, 1976, p.190-195.

TSICHRITIZIS, D.C.; BERNSTEIN, P.A. *Operating systems*. New York, N.Y. Academic Press, 1974.

WONG, C.K. Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems. *Computing Surveys*, 12(2): 167-178, June 1980.

YOUNG, M.S. Winchester/Floppy controller eases disk interfacing. *Computer Design*, 23(12): 129-137, Oct. 15, 1984.

YOUNG, M.S. Controllers wring peak performance out of disk drives. *Computer Design*, 24(5): 119-126, May 1985.

APÊNDICE A

PSEUDOCÓDIGO DAS PRIMITIVAS

Apresenta-se o pseudocódigo das primitivas de configuração e das primitivas de manipulação de arquivos.

A.1 - PRIMITIVAS DE CONFIGURAÇÃO

A.1.1 - PRIMITIVA DE DEFINIÇÃO DA TABELA DE CONTROLE DE DISCO

```
procedimento def_tcd (disp, ntrl, tam_reg, tam_set, tam_bl)

inicio

  reserve espaco para a T.C.D (na memoria);

  calcule o numero de registros/setor;

  calcule o numero de setores/bloco;

  calcule o numero de setores/trilha;

  armazene os valores na T.C.D.;

  copie a T.C.D para o disco, utilizando as rotinas de E/S
  do sistema operacional;
  sinalize que ja' foi executada.

fim
```

A.1.2 - PRIMITIVA DE DEFINIÇÃO DOS PARÂMETROS DE TRANSFERÊNCIA DE DADOS

```
procedimento def_ptd (nbuf, tam_buf, nred, nbe)
-----------

inicio
------

  reserve espaco para os "buffers" (nbuf x tam_buf);

  crie a lista de "buffers" livres;

  inicialize os cabecalhos dos "buffers";

  armazene as variaveis "nred" e "nbe";

  sinalize que ja' foi executada.

fim
---
```

A.1.3 - PRIMITIVA DE CRIAÇÃO DE DIRETÓRIO

```
procedimento cria_dir (disp, narq, nativ, nbpr, nseg, nprot)
------------

inicio
------

  crie o arquivo diretorio no dispositivo "disp" com "narq" entradas,
  utilizando as rotinas de E/S do sistema operacional;
  reserve espaco na memoria para o diretorio com "nativ" entradas;
  crie o diretorio na memoria com "nativ" entradas;
  reserve espaco na memoria para "nativ" descritores de arquivo;
  crie o vetor de bits no dispositivo "disp";
  inicialize o vetor de bits com todos os blocos livres (bits = 0);
  armazene as variaveis "nbpr", "nseg" e "nprot";
  sinalize que ja' foi executada.

fim
---
```

A.1.4 - PRIMITIVA DE INICIALIZAÇÃO DO MONITOR DE DISCO

```
procedimento inic_MD
-------------

inicio
------

  verifique se todos os parametros foram configurados;

  se foram entao
  --       -----

    inicialize as variaveis necessarias;

    habilite a fase de execucao;

  senao
  -----

    msg_erro ("monitor nao-configurado").

fim
---
```

A.2 - PRIMITIVAS DE MANIPULAÇÃO

A.2.1 - PRIMITIVA DE CRIAÇÃO DE ARQUIVO

```
procedimento cria_arq (nome_arq, disp, contr_acesso, id_proc)
------------

inicio
------

  procure "nome_arq" no diretorio corrente (memoria);

  se encontrou entao
  --          -----
    chame msg_erro("arquivo ja' existe");
    -----
  senao
  -----
% Busca do diretorio no disco:

    a = disp;

    b = numero dos setores onde se localiza o diretorio;

    c = numero das trilhas onde se localiza o diretorio;

    d = endereco de transferencia para o diretorio;


    chame leit_set (a, b, c, d);  % leia todo o diretorio;
    -----

    procure "nome_arq" no diretorio (memoria);

    se encontrou entao
    --          -----
      chame msg_erro ("arquivo ja' existe");
      -----
    senao verifique o numero maximo de arquivos "narq";
    -----
      se o numero dos arquivos atual > narq entao
      --                                 -----
        chame msg_erro ("espaco insuficiente");
        -----
      senao verifique o numero de arquivos ativos "nativ";
      -----
        se o numero dos arquivos ativos > nativ entao
        --
```

```
        chame msg_erro ("espaco insuficiente");
        -----
      senao
      -----
        id_arq = identificador para o novo arquivo;

        obtenha o endereco do descritor, definido pela primitiva
        "cria_dir";

        adicione a entrada no diretorio corrente;

% Montagem do descritor para o novo arquivo :

        chame monta_descr (id_arq, disp, contr_acesso, id_proc).
        -----

fim.
---
```

A.2.2 - PRIMITIVA DE REMOÇÃO DE ARQUIVO

```
procedimento rem_arq (nome_arq, disp, id_proc)
------------
inicio
------
  procure "nome_arq" no diretorio corrente (memoria);

  se nao o encontrar entao
  --                 -----

% Busca do diretorio no disco:

  a = disp;

  b = numero dos setores onde se localiza o diretorio;

  c = numero das trilhas onde se localiza o diretorio;

  d = endereco de transferencia para o diretorio;

  chame leit_set (a, b, c, d); leia todo o diretorio.
  -----
    procure "nome_arq" no diretorio (memoria);

    se nao o encontrar entao
    --                 -----
      chame msg_erro ("arquivo nao-encontrado");
      -----
  obtenha "id_arq" para o arquivo (diretorio corrente).

% Busca do descritor no disco:

  a = disp;

  b = numero dos setores onde se localiza o descritor;

  c = numero das trilhas onde se localiza o descritor;

  d = endereco de transferencia para o descritor;

  chame leit_set (a, b, c, d); leia todo o descritor.
  -----
```

```
% Alteracao dos valores do descritor:

  a = id_arq     % identificador do arquivo;

  b = disp       % identificador do disco;

  c = posicao do contador de processos que acessam o arquivo;

  d = decr       % decrementar contador de processos;

  e = nao importa

  chame alt_descr (a, b, c, d, e)
  -----

% Teste do contador de processos

  se o contador = 0 entao
  --                -----

    retire a entrada deste arquivo do diretorio corrente;

    libere os registros pre'-alocados e os usados pelo arquivo,
     atualizando o vetor de bits da memoria.

% Atualizacao do descritor do arquivo no disco:

    a = disp;

    b = numero dos setores onde se localiza o descritor;

    c = numero das trilhas onde se localiza o descritor;

    d = endereco do descritor na memoria;

    chame escr_set (a, b, c, d);  % escreva todo o descritor
    -----

% Atualize o diretorio do disco:

    a = disp;

    b = numero dos setores onde se localiza o diretorio;

    c = numero das trilhas onde se localiza o diretorio;

    d = endereco do diretorio na memoria;

    chame escr_set (a, b, c, d);  % escreva todo o diretorio.
    -----
```

```
% Atualize o vetor de bits do disco:
    a = disp;
    b = numero dos setores onde se localiza o vetor de bits;
    c = numero das trilhas onde se localiza o vetor de bits;
    d = endereco do vetor de bits na memoria;
  chame escr_set (a, b, c, d);  % escreva todo o vetor de bits
  -----
fim.
---
```

A.2.3 - PRIMITIVA DE ABERTURA DE ARQUIVO

```
procedimento abre_arq (nome_arq, disp, tipo_acesso, id_proc)
------------
inicio
------
  procure "nome_arq" no diretorio corrente (memoria);

  se nao o encontrar entao
  --                 -----

% Busca do diretorio no disco:

    a = disp;

    b = numero dos setores onde se localiza o diretorio;

    c = numero das trilhas onde se localiza o diretorio;

    d = endereco de transferencia para o diretorio;

    chame leit-set (a,b,c,d); leia todo o diretorio
    -----
    se nao o encontrar entao
    --                 -----
      chame msg_erro ("arquivo nao-encontrado");
      -----
    obtenha o "id_arq" para o arquivo (diretorio corrente);

% Busca do descritor no disco:

  a = disp;

  b = numero dos setores onde se localiza o descritor;

  c = numero das trilhas onde se localiza o descritor;

  d = endereco de transferencia para o descritor;

% Verificacao da permissao e do tipo de acesso desejado:

    se o processo nao tem permissao ou o tipo e invalido entao
    --                                                   -----
      chame msg_erro ("acesso invalido").
      -----
```

```
% Verifique se o arquivo esta aberto:

    se o arquivo nao esta' aberto entao
    --                          -----

% Zere o ponteiro de leitura e de escrita no descritor:

      a = id_arq        % identificador do arquivo;

      b = disp          % identificador do disco;

      c = posicao do ponteiro de leitura;

      d = subs          % substituicao de parametro;

      e = 0             % novo valor do parametro;

      chame alt_descr (a, b, c, d, e);
      -----

      a = id_arq        % identificador do arquivo;

      b = disp          % identificador do disco;

      c = posicao do ponteiro de escrita;

      d = subs          % substituicao de parametro;

      e = 0             % novo valor do parametro;

      chame alt_descr (a, b, c, d, e);
      -----
      obtenha endereco da tabela de registros logicos do arquivo.
fim
---
```

A.2.4 - PRIMITIVA DE FECHAMENTO DE ARQUIVO

```
procedimento fech_arq (nome_arq, disp, id_proc)
------------
inicio
------
  procure "nome_arq" no diretorio corrente (memoria)

  se nao o encontrar entao
  --                 -----

% Busca do diretorio no disco:

    a = disp;

    b = numero dos setores onde se localiza o diretorio;

    c = numero das trilhas onde se localiza o diretorio;

    d = endereco de transferencia para o diretorio;

    chame leit-set (a,b,c,d); leia todo o diretorio
    -----
    se nao o encontrar entao
    --                 -----
      chame msg_erro ("arquivo nao-encontrado")
      -----
    senao
    -----
      chame msg_erro ("arquivo fechado");
      -----
    obtenha "id_arq" para o arquivo (diretorio corrente);

% Busca do descritor do disco:

    a = disp;

    b = numero dos setores onde se localiza o descritor;

    c = numero das trilhas onde se localiza o descritor;

    d = endereco de transferencia para o descritor.

% Alteracao dos valores do descritor:
```

```
    a = id_arq;      % identificador
    b = disp;
    c = posicao do contador de processos que acessam o arquivo;
    d = decr         % operacao de decrementar
    e = nao importa;
    chame alt_descr (a, b, c, d, e);
    -----

% Teste do contador de processos
  se o contador for = 0 entao
  --                  -----
    libere os blocos pre-alocados.

% Atualizacao do descritor do arquivo no disco:
    a = disp;
    b = numero dos setores onde se localiza o descritor;
    c = numero das trilhas onde se localiza o descritor;
    d = endereco do descritor na memoria;
    chame escr_set (a, b, c, d);  % escreva todo o descritor
    -----

% Atualizacao do diretorio do disco:
    a = disp;
    b = numero dos setores onde se localiza o diretorio;
    c = numero das trilhas onde se localiza o diretorio;
    d = endereco do diretorio na memoria;
    chame escr_set (a, b, c, d);  % escreva todo o diretorio
    -----

fim
---
```

A.2.5 - PRIMITIVA DE MONTAGEM DE NOVO DESCRITOR

```
procedimento monta_descr (id_arq, disp, contr_acesso, id_proc)
------------
inicio
------

  obtenha endereco do (novo) descritor no diretorio do disco "disp"
  (memoria);
  armazene "id_arq" no descritor;
  armazene "id_proc" no descritor;
  crie a lista de controle de acesso com "nprot" entradas;
  armazene o ponteiro para a lista de controle de acesso;
  preencha a lista de controle de acesso com "contr_acesso";
  zere o contador de posicao de escrita;
  zere o contador de posicao de leitura;
  crie a lista de registros logicos do arquivo com "nseg" entradas;
  armazene o ponteiro para a lista de registros logicos;
  zere o contador de processos que acessam o arquivo;
  zere o contador de registros pre-alocados;
  zere o contador de registros do arquivo;
  armazene a data de criacao do arquivo (caso o sistema suporte)

fim
---
```

A.2.6 - PRIMITIVA DE ALTERAÇÃO DE PARÂMETRO DO DESCRITOR

```
procedimento alt_descr (id_arq, disp, desloc, oper, novo_val)
------------
inicio
------
  localize o descritor do arquivo na memoria, a partir de
  "id_arq" e "disp"
  localize o parametro a ser alterado no descritor, a partir de
   "desloc"
  identifique a operacao desejada "oper";
  caso oper = inc faca
  ----          ----
    parametro = parametro + 1
  caso oper = dec faca
  ----          ----
    parametro = parametro - 1
  caso oper = subs faca
  ----          ----
    parametro = novo_val
fim
---
```

A.2.7 - PRIMITIVA DE ESCRITA DE REGISTRO LÓGICO (COM PRÉ-ALOCAÇÃO)

```
procedimento escr_reg (id_arq, disp, nreg, id_proc)
------------
inicio
------
  localize o descritor do arquivo na memoria, a partir de
  "id_arq" e "disp"
  verifique o indicador de escrita
  se o indicador for = 1 entao
  --                     -----
    msg_erro ("acesso concorrente para escrita")
  verifique o contador de registros pre-alocados do arquivo;
  se o contador for = 0 entao
  --                    -----
    consulte o vetor de bits do disco "disp" (memoria);
    se existe espaco livre entao
    --                    -----
      marque o bloco como alocado;
      pre-aloque "nbpr" blocos contiguos;
      atualize o contador de registros do arquivo;
      atualize o contador de registros pre-alocados;
    senao chame msg_erro ("espaco insuficiente");
    -----

% Mapeamento logico x fisico (atraves da T.P.D):
    a = disp;
    b = numero do setor fisico (atraves de "nreg");
    c = numero da trilha desejada;
    d = endereco de transferencia (atraves de "id_proc")
```

```
% Escrita no disco:

    chame escr_set (a, b, c, d);
    -----
    se ocorrer erro entao
    --                 -----
      chame msg_erro ("erro de escrita").
      -----

fim
---
```

A.2.8 - PRIMITIVA DE LEITURA DE REGISTRO LÓGICO

```
procedimento leit_reg (id_arq, disp, nreg, id_proc)
------------
inicio
------
  localize o descritor do arquivo na memoria, a partir de "id_arq" e
   "disp".

% Mapeamento logico-fisico (atraves da T.P.D):

  a = disp;
  b = numero do setor fisico (atraves de "nreg");
  c = numero da trilha desejada;
  d = endereco de transferencia (atraves de "id_proc");

% Leitura do disco:

  chame leit_set (a, b, c, d);
  -----
    se ocorrer erro entao
    --                -----
      chame msg_erro ("erro de leitura").
      -----
fim
---
```

A.2.9 - PRIMITIVA DE LEITURA DE SETOR FÍSICO

```
procedimento leit_set (disp, set, trilh, end);
------------

inicio
------

  a = disp  ;  % identificacao do disco;

  b = 1     ;  % tipo da operacao = leitura;

  c = set   ;  % numero do setor;

  d = trilh ;  % numero da trilha;

  e = end   ;  % endereco para a transferencia dos dados;

  chame es_buf (a, b, c, d, e);
  -----

  se ocorrer erro entao
  --           -----
    retorne o codigo do erro

  senao
  -----

    retorne 0

fim
---
```

A.2.10 - PRIMITIVA DE ESCRITA DE SETOR FÍSICO

```
procedimento escr_set (disp, set, trilh, end);
------------

inicio
------

  a = disp  ;  % identificacao do disco;

  b = e     ;  % tipo da operacao = escrita;

  c = set   ;  % numero do setor;

  d = trilh ;  % numero da trilha;

  e = end   ;  % endereco para transferencia dos dados;

  chame es_buf (a, b, c, d, e);
  -----
  se ocorrer erro entao
  --                 -----
    retorne o codigo de erro

  senao
  -----
    retorne 0
fim
---
```

APÊNDICE B

PSEUDOCÓDIGO DAS ROTINAS DE OTIMIZAÇÃO

Apresenta-se o pseudocódigo das rotinas que implementam o algoritmo de otimização das transferências de dados e o de minimização dos movimentos do braço do acionador de discos (escalonamento). Tais rotinas são parte integrante do nível 2 - C.E.S., baseadas em Mckeon (1985).

B.1 - ROTINA DE E/S QUE USA "BUFFERS"

```
Nome: es_buf
----

Descricao: faz a transferencia de um bloco de dados de um
---------
           periferico para os "buffers" do sistema "cache" ou
           vice-versa.

Entradas: disp = identificacao do periferico;
--------
          l/e = tipo da operacao ( leitura ou escrita );
          set,trilh = identificacao do setor desejado;
          end = endereco para transferencia de dados.

Saidas:  nenhuma
------

procedimento es_buf (disp, l/e, set, trilh, end)
-----------

inicio
------

  incremente o contador de chamadas 'a rotina es_buf;
  se o contador e' multiplo de "nred" entao
  --                             -----
    aponte para o primeiro "buffer";
    para todos os buffers faca
    ----                 ----
      reduza XBFREQ pela metade;
      aponte para o proximo "buffer";
    volte para o "loop"
    -----

% Verifique se o "buffer" desejado esta' na memoria
  chame proc_buf (disp, set, trilh)
  -----
```

```
se achar o "buffer" desejado entao
--                          -----

  incremente XDFREQ;  % frequencia de acesso;

  atualize XDULT;     % param. ultimo acesso;

  se l/e=leitura entao
  --             -----

% Transferencia dos dados do "buffer" para a memoria;

    chame mov_dad (buf, end);
    -----

    retorne;

  se l/e=escrita entao
  --             -----

% Transferencia dos dados da memoria para o "buffer";

    chame mov_dad (end, buf);
    -----

    marque o "buffer" como do tipo escrita;

    retorne;

% Caso nao encontrar o "buffer" desejado;

  chame aloc_buf ()
  -----

  preencha o cabecalho do "buffer";

  se l/e=escrita entao
  --             -----
    chame mov_dad (end, buf)
    -----
    marque o "buffer" como do tipo escrita

  senao
  -----

% Realizacao da leitura antecipada

    enquanto nao for lido um bloco fisico faca;
    --------
      chame es_disp (disp, l, set, trilh,end);
      -----
```

```
  calcule o proximo setor, trilh;

volte para o "loop";
-----
    se ocorrer erro entao
    --             -----
      chame lib_buf()    % libere o "buffer"
      -----
      retorne o codigo de erro

    senao

      marque o "buffer" como do tipo leitura

      volte para o inicio.
                   ------
```

B.2 - ROTINA DE ALOCAÇÃO DE "BUFFER"

```
Nome: aloc_buf
----

Descricao: alocá um "buffer" da memoria, procurando pelo que tiver a
---------
           menor frequencia de uso e que seja o menos usado

           recentemente (L.R.U).

Entradas: nenhuma
--------

Saidas: endereco do" buffer" alocado ou mensagem-de-erro.
------


Sub-rotina aloc_buf ()
----------

inicio
------

  verifique se existe "buffer" livre

  se existir entao
  --          -----
    retire um "buffer" da lista de "buffers" livres

    retorne (endereco do "buffer" livre)
    -------
  senao aponte para o primeiro "buffer"
  -----
    para todos os "buffers" faca
    ----                    ----
      se o "buffer" e' de leitura entao
      --                       -----
        incremente o numero de "buffers" de leitura

        se freq.uso < freq.uso do "buffer" anterior entao
        --                                       -----
          se o "buffer" foi menos acessado que o anterior entao
          --                                             -----
            guarde o endereco desse "buffer"

      aponte para o proximo "buffer"

    volte para o "loop"
    -----
```

```
  faca o numero de "buffers" de escrita= numero total "buffers" -
  ----
  numero de "buffers" de leitura

  se o numero "buffers" escrita >  "nbe" entao
  --                               -----

% transfira os "buffers" de escrita pendentes para disco

    chame escr_disc ()
    -----
  se achou "buffer" de leitura entao
  --

    chame lib_buf () % libera o "buffer" encontrado

    va para o inicio % tenta novamente
    --

  senao se escreveu no disco entao
  -----                      -----
        va para o inicio
        --
        senao chame msg_erro ("erro de alocacao").
        ----- -----
fim
---
```

B.3 - ROTINA DE PROCURA DE "BUFFER" DESEJADO

```
Nome: proc_buf
----

Descricao: procura pelo "buffer" especificado, entre todos os
---------
           "buffers" da memoria.

Entradas: parametros
--------
          disp = periferico ao qual pertence o "buffer";

          set = numero do setor;

          trilh = numero da trilha.

Saidas: endereco do "buffer" se encontrado, ou 0 se nao encontrado
------

Sub-rotina proc_buf (disp, set, trilh);
----------

inicio
------
  aponte para o primeiro "buffer" da memoria

  para todos os "buffers" faca
  ----                    ----
    verifique o tipo do "buffer"

    se nao e' "buffer" livre entao
    --                     -----
      verifique se (set, trilh) sao iguais

      se forem entao
      --       -----
        retorne (endereco do "buffer")
        -------
    aponte para o proximo "buffer"

  volte para o "loop"
  -----
  retorne 0
  -------
fim
---
```

B.4 - ROTINA DE ESCRITA NO DISCO COM ESCALONAMENTO

```
Nome: escr_disc
----

Descricao: transfere para o disco todos os "buffers" de escrita
---------  pendentes, convertendo-os para buffers do tipo leitura.
           A escrita e' feita de maneira a minimizar o movimento do
           braco do disco, isto e', a partir do periferico com maior
           identificador e do setor de menor identificagr
           (conjunto: trilh, set).

Entradas: nenhuma
--------

Saidas: numero de "buffers" escritos "nescr"
------


Sub-rotina escr_disc ()
-----------

inicio
------
  inicializacao: nescr=0 % contador de "buffers" escritos no disco

  chame buf_escr () % procura o "buffer" a ser escrito
  -----

% Chame a rotina de E/S do Sistema Operacional:

  enquanto houver "buffer" de escrita faca
  --------                          ----
    chame es_disp (disp, e, set, trilh, end)
    -----
    mude o tipo do "buffer" para leitura

    nescr = nescr + 1 % incrementa o contador

  volte para "loop"
  -----
  retorne (nescr)
  -------
fim
---
```

B.5 - ROTINA DE PROCURA DE "BUFFER" DE ESCRITA

```
Nome: buf_escr
----

Descricao: procura entre todos os "buffers" aquele de escrita que
---------  pertence ao periferico com maior identificador e ao setor
           de menor identificador (conjunto trilh, set).

Entradas: nenhuma
-------

Saidas: endereco do "buffer" de escrita, ou 0 se nao existe um
------  "buffer" de escrita

Sub-rotina buf_escr ()
----------

inicio
------

  inicialize as variaveis: disp=0, end=0, (trilh, set)=max
  aponte para o primeiro "buffer" da cadeia de "buffers"
  para todos os "buffers" faca
  ----                    ----
    se e' "buffer" do tipo escrita entao
    --
      se disp < disp. ao qual pertence o "buffer" entao
      --
        disp = disp. ao qual pertence o "buffer"
        (trilh, set) = (trilh, set) do "buffer";
        end = endereco do "buffer"
      senao
      -----
        se disp = disp.ao qual pertence o "buffer" entao
        --                                        -----
        se (trilh, set) > (trilh, set) do "buffer" entao
        --                                         -----
        (trilh, set) = (trilh, set) do "buffer"
        end = endereco do "buffer".
        aponte para o proximo "buffer"
      volte para o "loop"
      -----
      retorne (end)
      -------
  fim
  ---
```

B.6 - ROTINA DE LIBERAÇÃO DE "BUFFERS"

```
Nome: lib_buf
----

Descricao: libera um "buffer", devolvendo-o a lista de "buffers"
---------
           livres.

Entradas: parametro
--------
          buf = ponteiro do "buffer" a ser liberado

Saidas: nenhuma
------

Sub-rotina lib_buf (buf)
----------

inicio
------

  aponte para a area de cabecalho do "buffer"

  marque o "buffer" como livre e

  coloque na lista de "buffers" livres

  atualize o apontador da lista de "buffers" livres

fim
---
```

CONSELHO NACIONAL DE DESENVOLVIMENTO CIENTÍFICO E TECNOLÓGICO
INSTITUTO DE PESQUISAS ESPACIAIS

PROPOSTA PARA PUBLICAÇÃO

DATA: 03.07.86

IDENTIFICAÇÃO

TÍTULO

Monitor de Disco de alto desempenho para aplicações em tempo real

AUTOR

Maurício Macedo de Faria

ORIENTADOR

Ricardo Corrêa Oliveira Martins

CO-ORIENTADOR

CURSO		DATA DE DEFESA
ECO/SDA	☐ TESE ☒ DISSERTAÇÃO	18.06.86

REVISÃO TÉCNICA FINAL

CONCLUÍDA EM: 03/07/86 — Ricardo — ORIENTADOR

REVISÃO DE LINGUAGEM

Nº: 295 PRIORIDADE: 2 DATA: 4.7.86

POR: Rousa Maria Dias Bicudo

OBSERVAÇÕES:

31.01.87 — DATA

Rousa M D Bicudo — ASSINATURA

EM CONDIÇÕES DE PUBLICAÇÃO EM: _____ _____ AUTOR

AUTORIZO A PUBLICAÇÃO: ☐ SIM ☐ NÃO

OBSERVAÇÕES:

_____ DATA — _____ DIRETOR

SECRETARIA EXECUTIVA

PUBLICAÇÃO: 4149-TDL/267 PÁGINAS: _____ ÚLTIMA PÁGINA: _____

CÓPIAS: _____ TIPO: _____ PREÇO: _____

OBSERVAÇÕES:

OBSERVAÇÕES E NOTAS

SUBSCREVÊ - LAS COM A DEVIDA RUBRICA