UM SERVIDOR DE ARQUIVOS PARA UMA REDE LOCAL

Maria Cristina Bastos Monteiro

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.)

Aprovado por

Eber Assis Schmitz

(Presidente)

Édil Severiano Tavares Fernandes

Luiz Antonio Carneiro da Cunha Couceiro

RIO DE JANEIRO, RJ-BRASIL

MARÇO DE 1984

Monteiro, Maria Cristina Bastos

Um Servidor de Arquivos para uma Rede Local (Rio de Janeiro) 1984

VIII, 120p. 29,7cm (COPPE-UFRJ, M.Sc., Engenharia de Siste - temas, 1984)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Servidores de Arquivos

I. COPPE/UFRJ II. Título (série)

Agradeço aos engenheiros da CETUS INFORMÁTICA S/A., em especial a Marcio Lannes Duarte de Souza, pela sua participação no projeto.

A Rosilene Victoria Alvares, agradeço os trabalhos de datilografia.

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

UM SERVIDOR DE ARQUIVOS PARA UMA REDE LOCAL

Maria Cristina Bastos Monteiro

março de 1984

Orientador: Eber Assis Schmitz

Programa: Engenharia de Sistemas e Computação

Este trabalho descreve o projeto de um servidor de arquivos destinado a operar em um sistema distribuído, integrado por uma rede local. Trata do compartilhamento físico e lógico da memória de massa numa comunidade heterogênea de usuários (diferentes computadores com diferentes sistemas operacionais).

A estrutura adotada define um serviço de gerenciamento de arquivos apoiado no subsistema de transporte da rede utilizada. As principais características do referido serviço são:

- organização estrutural multi-nível para os arquivos
- primitivas de manipulação lógica dos dados visando assegurar compatibilidade funcional com os diferentes sistemas usuários
- aspectos de proteção incluindo: autenticação de usuários, privacidade da informação; e manutenção da integridade dos dados em caso de acesso concorrente ou falhas no sistema.

O trabalho realizado envolveu etapas de especificação, definição da arquitetura e implementação do sistema. Algumas medidas foram também realizadas visando avaliar parâmetros de projeto tais como memória cache, acesso indireto aos arquivos, e gravações atômicas.

Baseado neste trabalho está sendo elaborada uma versão industrial do servidor de arquivos a ser produzida e comercializada no decorrer deste ano.

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

UM SERVIDOR DE ARQUIVOS PARA UMA REDE LOCAL

Maria Cristina Bastos Monteiro

march, 1984

Chairman: Eber Assis Schmitz

Department: Engenharia de Sistemas e Computação

This thesis is concerned with the design of a File Server as an element of a distributed system based on a Local Area Network. It addresses the problem of sharing of mass memory among heterogeneous users - different computers and operating systems.

The file server is composed of a file management service which uses the transport subsystem of a Local Area Network. The main features of the service above are:

- multi-level file structure
- data mapulation primitives which enables functional compatibility among different systems.
- protection and privacy; maintenance of data integrity in cases of concurrent access or hardware failures.

This work involves the phases of: specification, architecture and system implementation. Some measurements were made for the evaluation of some system parameters such as: cache memory, file access and atomic transactions.

Based on this work, a commercial version of the File Server is in preparation.

ÍNDICE

CAPÍTULO I - INTRODUÇÃO

1.1 - Redes

1.2 - Redes Locais

1.3 - Servidores

1.4 - O Servidor de Arquivos

1.4.1 - Posicionamento do Problema

1.4.2 - Restrições

1.5 - Trabalhos Relacionados

1.6 - Plano da Tese

CAPÍTULO II - O SERVIDOR DE ARQUIVOS NUM AMBIENTE DISTRIBUÍDO

2.1 - Os Subsistemas de Comunicação em um Ambiente de Processametno Distribuído

2.2 - A Arquitetura Adotada pelo SA

2.3 - A Cooperação entre os Componentes de um Sistema Distribuído

2.3.1 - Compatibilidade Usuário-Servidor

2.3.2 - A Compatibilidade Usuário-Usuário

CAPÍTULO III - ESTRUTURA BÁSICA DO SISTEMA

3.1 - O Aspecto Estrutural da Memória de Massa

3.1.1 - Apresentação da Estrutura

3.1.2 - Identificação dos Arquivos no SA

3.1.3 - A Estrutura Interna de um Arquivo

3.1.4 - Alocação de Espaço Físico para um Arquivo
3.1.5 - Administração do Espaço Livre
3.1.6 - Overhead de Memória Imposto pela Organização Estrutural
3.2 - Manipulação Lógica dos Objetos
3.2.1 - O Gerenciamento da Memória de Massa
3.2.2 - Como Referenciar um Arquivo
3.2.3 - Proteção da Informação
3.2.4 - Os Direitos de Acesso
3.2.5 - Os Controles de Acesso
3.2.6 - Manutenção da Integridade dos Dados
3.3 - As Primitivas de Comunicação com o SGA
3.3.1 - A Escolha das Primitivas
3.3.2 - A Descrição das Primitivas
3.3.3 - Um Exemplo de Sessão

CAPÍTULO IV - IMPLEMENTAÇÃO

4.1 - Descrição Geral
4.2 - A Definição das Principais Constantes
4.3 - A Definição dos Principais Tipos Simples
4.4 - A Definição dos Principais Tipos Estruturados
4.5 - A Definição das Principais Variáveis Globais

4.6 - As Rotinas de Manipulação de Objetos

4.7 - As Funções de Serviço

4.8 - O Programa Principal

4.9 - O Gerenciamento dos Processos

CAPÍTULO I

INTRODUÇÃO

1.1- Redes

A queda no custo do processamento da informação estimulou efetivamente o uso da computação distribuída: tarefas que antes eram atribuídas a um único computador de capacidade limitada tendem agora a ser divididas entre vários computadores de menor custo situados, de forma geral, em diferentes localidades. Na sua forma geral um sistema distribuído é formado por um conjunto de elementos computacionais, geograficamente dispersos, interagindo através de um mecanismo de comunicação e cooperando entre si na execução de tarefas globais (figura I.1).

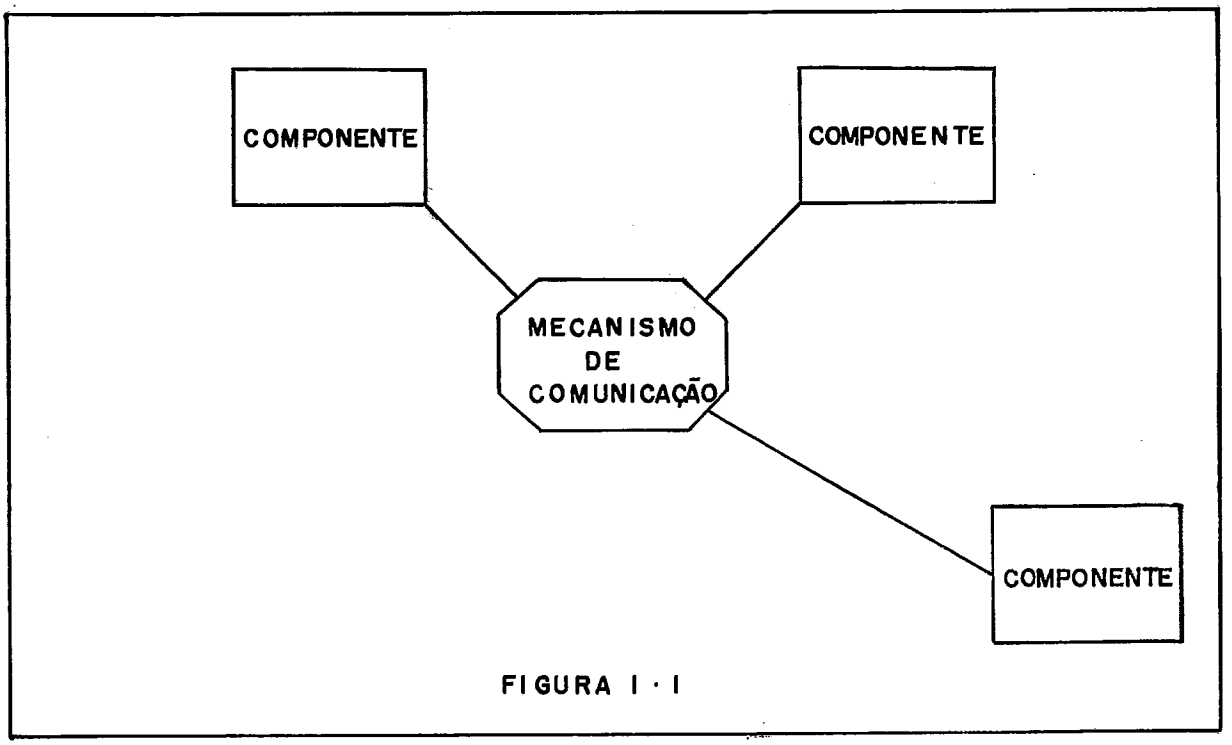


FIGURA I·1

Os sistemas distribuídos apresentam sobre os centralizados as seguintes vantagens:

. Disponibilidade

A falha de um elemento não implica na paralização das tarefas executadas pelos demais mas apenas reduz a capacidade de processamento do conjunto (degradação suave).

. Versatilidade

Os diferentes componentes do sistema podem se alternar na realização de tarefas, viabilizando transferência da atividade executada por um elemento em pane para um outro elemento (redundância).

. Otimização

Em lugar de um elemento central encarregado das diferentes atividades envolvidas no sistema, os diferentes componentes podem se especializar em tarefas particulares. Esta característica simplifica o projeto do componente , reduzindo custos de implementação, erros de software e custos de manutenção (especialização).

As redes de computadores |Bochmann, 1983| vêm resolver o problema da comunicação entre os elementos do sistema distribuído, permitindo o acesso remoto aos recursos disponíveis no conjunto. O compartilhamento de um banco de dados entre os diversos usuários de uma rede é um exemplo típico da vantagem obtida com esta interligação.

A primeira rede de computadores, a ARPANET, entrou em operação no início da década de 70 interligando computadores de instituições de ensino e pesquisa nos Estados Unidos. Experiência análoga foi realizada na França com a rede CYCLADES. Ambas utilizavam o conceito de "comutação de pacotes" segundo o qual mensagens são trocadas entre os elementos sob a forma de

pacotes de dados de tamanho padrão.

1.2- Redes Locais

Em contraste com esses sistemas envolvendo comunicações a longa distância, começaram a surgir nos anos 70 redes privadas interligando elementos de computação em uma área geograficamente limitada (um ou dois quilômetros) conhecidos como redes locais |Bochmann, 1983|. Existem tecnologias que, a essa distância, permitem a transmissão a uma alta taxa de transferência (de 500K bps a 15M bps) com alto grau de confiabilidade e relativo baixo custo.

O projeto de uma rede local deve levar em conta dois aspectos fundamentais:

- Descentralização

  O esquema de interligação não deve incluir elementos cuja falha afete a comunicação entre terceiros. Em outras palavras, num diálogo não há participação de nenhum outro elemento a não ser os diretamente envolvidos na conversação. Em caso de pane apenas o elemento afetado perderia a capacidade de se comunicar com os demais.

- Modularidade

  O custo na introdução de um novo elemento é praticamente igual ao valor do elemento, garantindo portanto uma excelente capacidade de expansão.

Existem dois parâmetros básicos de projeto que influenciam os graus de modularidade e descentralização do sistema: a topologia que é a maneira pela qual os elementos se interconectam fisicamente para formar a rede e o método de acesso que é a política segundo a qual os elementos acessam a via de

interconexão.

As três topologias principais são: a estrela, o anel e a barra (figura I.2).

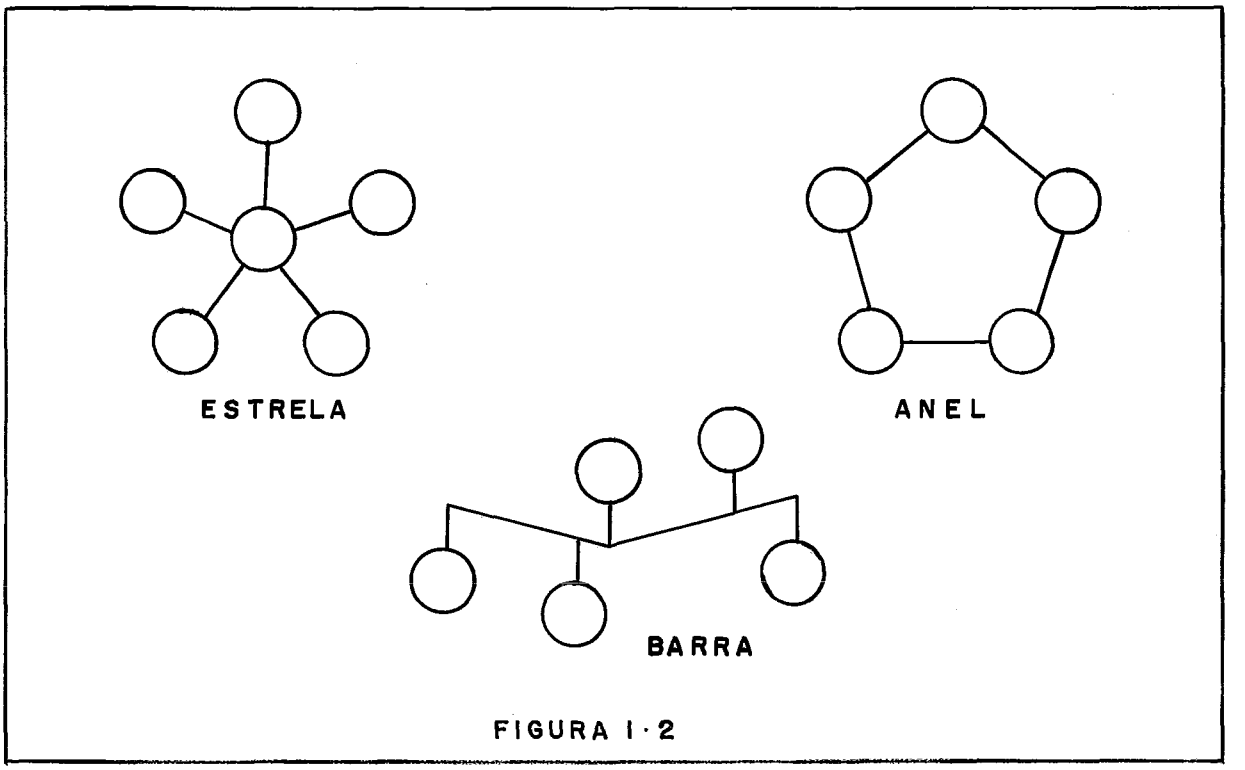


FIGURA I·2

Na topologia em estrela, todos os elementos se conectam a um elemento central que controla todas as comunicações. É um esquema de baixo grau de descentralização e modularidade.

Na topologia em anel, as informações trocadas entre dois elementos quaisquer passam pelos elementos intermediários onde são retransmitidas até atingirem o elemento destino.

Na topologia em barra, as informações transmitidas se difundem pela rede atingindo todos os elementos.

Os dois principais métodos de acesso são o de passagem de mando e o randômico. No primeiro existe uma mensagem de controle (o mando) que é passada de elemento para elemento da rede. Apenas o elemento que possui o mando pode fazer uso da via de interconexão, os demais limitam-se a ouvir. No método

de acesso randômico, o acesso à via é considerado independentemente por cada elemento. A implementação que consagrou o seu uso foi a rede Ethernet, |Metcalfe, 1976| lançada em 1976 pela Xerox, sendo o método conhecido como CSMA-CD (Carrier Sense Multiple Access - Collision Detection). Neste método, quando alguém quer transmitir segue o seguinte procedimento: verifica se existe alguém transmitindo ("carrier sense"); em caso negativo, a transmissão é iniciada; se durante a transmissão ocorrer interferência ("collision detection"), a transmissão é interrompida para ser reiniciada após um tempo randômico.

Baseado no "Modelo de Referência para a Interconexão de Sistemas Abertos" da ISO |OSI, 1980|, o IEEE gerou uma recomendação para redes locais |IEEE, 1982| sugerindo para acesso e topologia as seguintes combinações: CSMA-CD X bus, passagem de mando x bus e passagem de mando x anel.

O compartilhamento de recursos e informações é a motivação essencial para o surgimento das redes de computadores. Em particular, a rede local que serve a uma comunidade limitada de usuários dentro de uma empresa, tem interesse especial em compartilhar periféricos caros tais como discos e impressoras. Este trabalho define o elemento que atua como interface inteligente entre a rede e um dispositivo de memória de massa a ser compartilhado entre diversos usuários: o servidor de arquivos.

1.3- Servidores

Dentre os elementos ligados a uma rede local aqueles que prestam um suporte especializado aos demais são denominados servidores |Wilkes, 1980; Souza, 1983; Schimtz, 1983|.

Os servidores distinguem-se dos demais elementos pelo software e/ou hardware específicos que contêm. Dentre os servidores típicos podemos citar:

. Servidor de Arquivos

Sua tarefa consiste em oferecer o compartilhamento físico e lógico de memória de massa entre os diver - sos usuários da rede fornecendo aos mesmos os serviços clássicos de um sistema de arquivos.

. Servidor de Impressão

Oferece aos usuários serviços tais como "spooling", formatação de textos e impressão de relatórios.

. Relógio de Tempo Real

Este servidor fornece a data e a hora quando requisitado.

. Servidor de Banco de Dados

Este servidor distingue-se do servidor de arquivos da mesma forma que um sistema de gerenciamento de banco de dados é funcionalmente diferente de um sistema de arquivos no contexto usual de processamento de dados.

Sua implementação consiste portanto em aliar, num processsador dedicado, um software de manipulação de dados a uma linguagem de acesso adequada.

A escolha dos servidores a serem utilizados num sistema integrado por uma rede local não é rígida nem o campo de ação de um determinado servidor é obrigatoriamente padronizado. Sempre que, num contexto particular, uma situação exigir que diversos usuários implantem em suas máquinas a mesma tarefa, é sinal que a criação de um servidor pode ser uma boa opção. Aliás, o número crescente de redes locais aumenta o interesse em diversos serviços que poderão ser prestados num ambiente de computação distribuída.

Por exemplo, a tarefa de edição pode ser entregue

aos computadores ligados à rede ou pode ser definida como uma tarefa do sistema. Neste caso, adota-se um servidor de edição que permitirá criar e editar arquivos sem a necessidade de alocação de computador . Tal política pode ser atraente, do ponto de vista econômico, num ambiente de processamento de textos ou gráficos.

1.4- O Servidor de Arquivos

1.4.1- Posicionamento do Problema

O servidor de arquivos (SA) definido neste trabalho opera num sistema distribuído composto de diversos elementos (computadores de todos os tipos, vídeos, impressoras...) interligados pela rede local CETUS (vide Anexo I).

O termo usuário, em relação ao SA, será usado aqui para representar uma pessoa utilizando um terminal, um programa aplicativo ou o próprio sistema operacional da máquina ligada à rede.

Os objetivos perseguidos pelo SA são os seguintes:

- permitir o compartilhamento físico de memória de massa e o compartilhamento lógico da mesma entre os diferentes usuários da rede (obedecer a uma política de compartilhamento);
- servir simultâneamente a sistemas de arquivos ("file systems") diferentes, isto é, suportados por sistemas operacionais diferentes;
- fornecer um sistema de arquivos completo que possa ser utilizado pelo usuário da rede no lugar do sistema próprio da sua máquina;

- garantir privacidade aos arquivos sob sua guarda, seguindo uma política de direitos de acesso que impeça a ação indesejada de usuários distraídos ou maliciosos;

- preservar a integridade dos arquivos sob sua guarda contra acessos concorrentes e falhas no sistema.

1.4.2- Restrições

Na prática, quanto mais amplos são os objetivos a serem alcançados, maiores são os comprometimentos do ponto de vista de complexidade de implementação, eficiência, custo e confiabilidade.

Dentro do espectro de funções realizadas por um sistema de arquivos convencional, caminhar do extremo mais simples ao mais complexo significa variar o grau de responsabilidade que se confere ao usuário. Quanto mais ampla for a concepção do SA maior a responsabilidade que se entrega ao mesmo e menores são portanto as atribuições que restam ao usuário.

Cada um dos extremos deste espectro representa um tipo de SA com facilidades, custos e desempenhos bem definidos. Segue-se a análise destas duas versões extremas.

i) Versão Simplificada (responsabilidade máxima para o usuário).

a) O SA é um dispositivo de memória de massa remoto, ligado através da rede, que o usuário enxerga como se fosse exclusivamente seu, isto é, não há compartilhamento.

b) O usuário é totalmente responsável pela estrutura imposta aos arquivos. O SA não interfere na dispo-

sição lógica da informação.

c) O SA tem como função básica gerenciar o espaço livre disponível no dispositivo controlando uma tabela de mapeamento dos blocos lógicos dos usuários em posições físicas (endereços) do dispositivo;

d) As primitivas para comunicação com o SA reduzem-se a:
   - ler n blocos localizados a partir do endereço lógico x
   - escrever n blocos a partir do endereço lógico x

ii) Versão Complexa (responsabilidade mínima para o usuário).

a) O SA é uma interface inteligente, remota, ligada através da rede, que o usuário enxerga como um gerenciador de arquivos dentro de um sistema de "time-sharing" e compartilhamento de informações;

b) O SA é totalmente responsável pela estrutura dos arquivos e operações realizadas sobre os mesmos, controlando diretórios, compartilhamento de arquivos, direitos de acesso, métodos de acesso, etc.;

c) O SA recebe do usuário pedidos de operações de alto nível cujos parâmetros são entre outros os nomes dos arquivos sob forma alfanumérica. Os comandos disponíveis são análogos aos disponíveis nas linguagens clássicas de alto nível (PASCAL, FORTRAN, etc.)

Do ponto de vista do usuário da rede local, aumentar a responsabilidade do SA pode parecer a solução mais atraente pois permite ao usuário reduzir as tarefas executadas em sua

máquina. Uma análise mais cuidadosa, no entanto, nos chama a atenção sobre a grande quantidade de software realizado sobre os sistemas operacionais destas máquinas que trazem embutidos sistemas de arquivos completos. Tentar transferir esta capacidade, já implantada em cada máquina, para o SA incorreria em repetir no servidor esforços já existentes nos equipamentos usuários (duplicação de esforços).

Podemos concluir que, para cumprir os objetivos fixados em 1.4.1, o SA deve permitir a conexão de usuários com níveis de responsabilidade diferentes.

1.5- Trabalhos Relacionados

O interesse crescente em fornecer serviços para sistemas distribuídos resultou em um número considerável de implementações para servidores de arquivos.

Na Universidade de Cambridge foi desenvolvido um servidor |Dion, 1980| para os usuários da sua rede em anel ("The Cambridge Digital Communications Ring"). Suas principais características são: alta velocidade de acesso a objetos; capacidade de realizar atualizações atômicas; controle de acesso baseado no conceito de "capabilities"; recolhimento automático de memória sem uso; e recuperação automática mediante falhas.

O Centro de Pesquisas da Xerox (Palo Alto) concebeu um serviço distribuído de arquivos |Sturgis, 1980|. A implementação foi realizada num conjunto de computadores, atuando como servidores, conectados a uma rede local e especialmente voltada para pesquisa de banco de dados. A distribuição de informação por entre diversos servidores, criando juntos a ilusão de um único sistema lógico, visou conferir flexibilidade ao dimensionamento da capacidade de memória e processamento. São fornecidas então primitivas especiais de sincronização entre os diver-

sos componentes . Este projeto também se preocupou com a consistência dos dados armazenados, empregando métodos para realização de escritas atômicas diante de falhas e utilizando o conceito de "memória estável" na implementação da redundância estrutural do sistema.

A Universidade de Carnegie - Mellon desenvolveu um sistema de arquivos centralizado |Robertson, 1980|, baseado no UNIX, destinado ao fornecimento de serviços para uma comunidade heterogênea de usuários de uma rede local genérica. Além dos serviços básicos de compartilhamento de arquivos, este projeto se preocupou particularmente com facilidades de arquivamento e flexibilidade na concepção dos direitos de acesso.

Por outro lado, surgem trabalhos como o desenvolvido para a rede local da Universidade da Califórnia |Rowe, 1982|,dedicados a incorporar protocolos de gerenciamento de arquivos em sistemas operacionais comerciais. No caso, foi realizada uma adaptação no sistema operacional UNIX |Thompson, 1974| de modo que programas já existentes pudessem acessar arquivos remotos sem modificação dos seus fontes.

1.6- Plano da Tese

No capítulo II trataremos da arquitetura de sistemas distribuídos observando, em particular, as funções dos subsistemas envolvidos. Esta análise procura apresentar a estrutura na sua forma hierárquica típica, discutindo os principais tipos de protocolos envolvidos. Procederemos então à análise do caso particular do SA dentro deste contexto, descrevendo a arquitetura adotada para o mesmo. Discute-se a seguir a cooperação entre o SA e os demais elementos da rede, analisando os requisitos de compatibilidade.

O capítulo III apresenta a estrutura básica do sistema de gerenciamento de arquivos (SGA). Para tal, são definidas inicialmente todas as estruturas de dados envolvidas e, a seguir, a manipulação lógica que o SA realiza sobre as mesmas. Neste capítulo definiremos portanto os serviços fornecidos pelo SGA.

O capítulo IV corresponde à implementação. As estruturas e funções descritas no capítulo anterior são realizadas sob a forma de algoritmos de alto nível.

Finalmente, no capítulo V, procederemos à análise de resultados práticos obtidos em laboratório.

As conclusões sobre os principais aspectos do trabalho são apresentadas no capítulo VI.

CAPÍTULO II

O SERVIDOR DE ARQUIVOS NUM AMBIENTE DISTRIBUÍDO

2.1- Os subsistemas de comunicação em um ambiente de processamento distribuído.

O problema do SA pode ser visto como um caso particular do problema mais geral de cooperação de processos em um ambiente de processamento distribuído. Envolve assim questões básicas de: comunicação entre os elementos do sistema, sincronização, e consistência de operações realizadas em diferentes pontos |Bochmann, 1983|.

Assim, dois processos que antes interagiam diretamente num sistema centralizado (figura II.1) agora se comunicam através de um subsistema de comunicação (SC) num sistema distribuído (figura II.2).

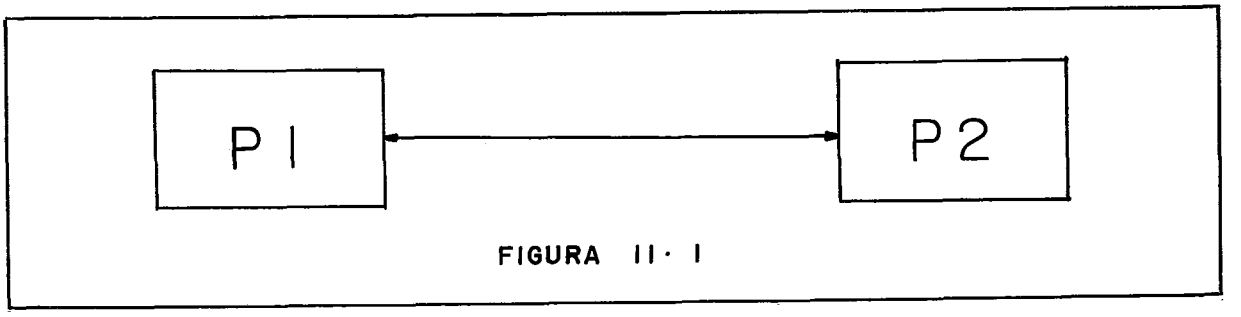


FIGURA II·1

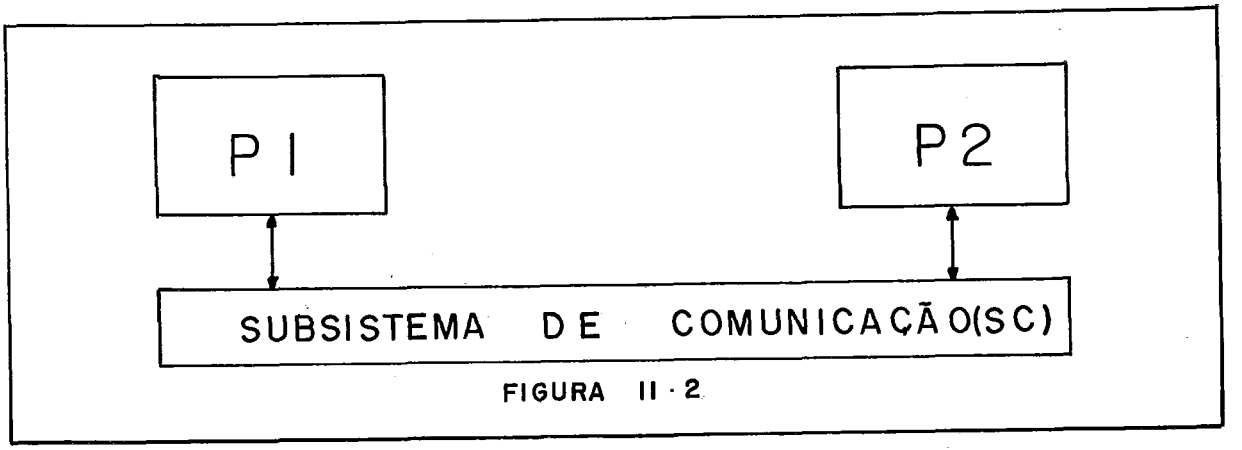


FIGURA II·2

O subsistema de comunicação é tão mais transparente quanto mais parecer aos processos que a sua comunicação com o SC é análoga à sua interação direta. Na prática, algumas características inerentes aos sistemas de comunicação são fatores que limitam essa transparência: erros de transmissão, delays, perda ou duplicação de mensagens, alteração na sequência das mensagens etc.

O SC , por sua vez, é usualmente decomposto em camadas segundo uma estrutura hierárquica modular conforme ilustra a figura II.3.

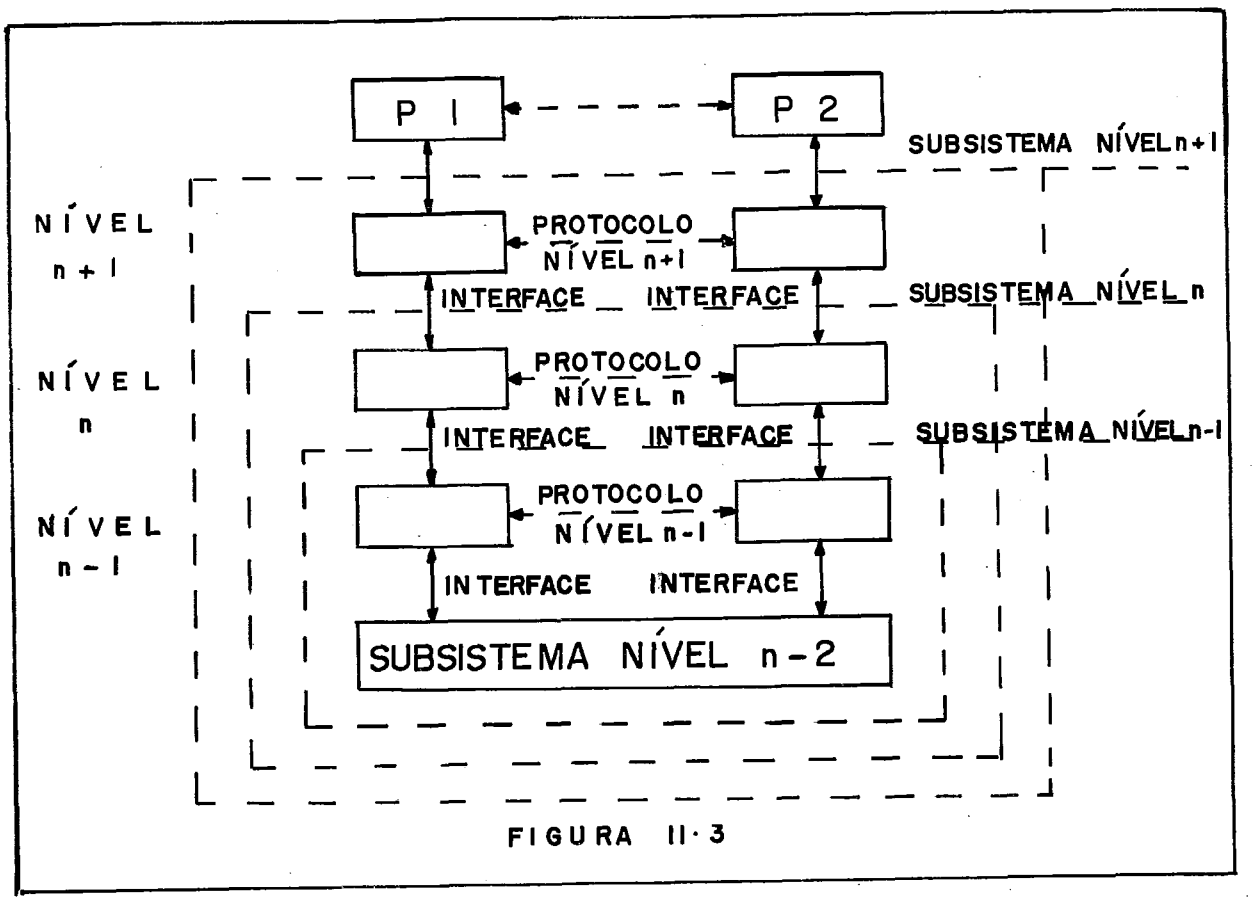


FIGURA II·3

Uma estrutura complexa é assim decomposta em camadas funcionalmente independentes. Cada camada implementa um conjunto de funções usando apenas os serviços oferecidos pela camada imediatamente inferior. As referidas funções geram por sua vez um conjunto de serviços que poderão ser acessados apenas pela camada imediatamente superior.

A interface entre dois níveis constitui-se de um conjunto de primitivas, denominadas primitivas de serviço, que definem os tipos de interação entre as referidas camadas. A especificação dos serviços fornecidos por uma camada inclui as primitivas de serviço e as possíveis seqüências de execução das mesmas.

Um subsistema de nível n é constituído pelas camadas de nível 1 a n. O seu comportamento é definido pelo serviço especificado para a camada de nível n.

Programas pertencentes ao mesmo nível n mas situadas em pontos diferentes do sistema distribuído necessitam se comunicar, passando dados e informações de controle através das camadas inferiores do sistema. Esta comunicação envolve um conjunto de mensagens e de regras conhecidas como protocolo de comunicação de nível n. Do ponto de vista do protocolo, a mensagem pode ser considerada como passando diretamente do nível localizado no equipamento transmissor para o nível correspondente no receptor, conforme sugerem as linhas pontilhadas na figura II.3.

A adoção desta disciplina hierárquica traz uma série de vantagens como: obriga o projetista a desenvolver um trabalho bem estruturado; favorece a especificação e documentação das interfaces entre as camadas; facilita a criação de padrões efetivos; possibilita a comunicação de sistemas concebidos em organizações distantes; permite a substituição de camadas intermediárias por outras equivalentes sem afetar o restante da estrutura, isto é, garante a evolução do sistema; e facilita a depuração de erros.

Baseado nesses conceitos, a ISO ("International Standards Organization") |OSI, 1980| elaborou um modelo de referência para a interligação de sistemas abertos, descrevendo

os serviços oferecidos pelos sete seguintes níveis:

- Físico

  Se refere aos meios reais de transmissão de bits através de um meio físico.

- Linha

  Garante que seqüências lógicas de mensagens sejam trocadas de maneira confiável através de um único canal físico.

- Rede

  Fornece canais lógicos capazes de transferir informação entre dois pontos extremos de uma rede de comunicação.

- Transporte

  Implementa o transporte confiável de mensagens entre dois pontos do sistema distribuído através de uma topologia arbitrária para a comunicação.

- Sessão

  Estabelece sessões para serem utilizadas em um diálogo de alto nível entre dois processos.

- Apresentação

  Fornece as transformações de formato das informações a serem transferidas.

- Aplicação

  Se refere à aplicação propriamente dita.

  A figura II.4 ilustra os níveis descritos.

Os quatro primeiros níveis constituem o subsistema de transporte, e fornecem ao nível superior uma interface independente da rede utilizada, caracterizada por primitivas de serviço tais como: conexão, desconexão, transmissão e recepção de mensagens: Os protocolos envolvidos nestes quatro primeiros

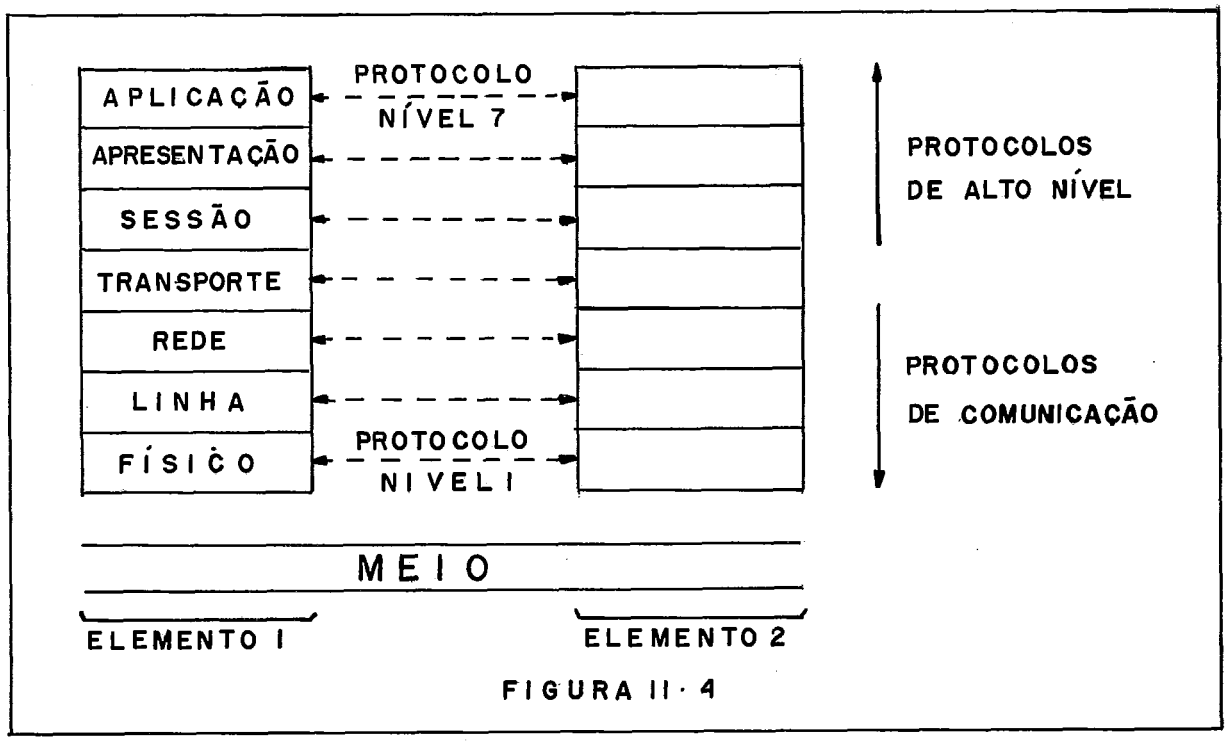


FIGURA II·4

níveis são por isso chamados de protocolos de comunicação.

Em contraste com os protocolos de comunicação, os protocolos acima do nível de transporte são denominados protocolos de alto nível |DAY 1979; GIEN, 1978|. Eles são ainda assunto de pesquisa e experiência não tendo sido ainda suficientemente analisados pelos órgãos de padronização sobretudo no que diz respeito a redes locais.

Os usuários dos protocolos de alto nível são atividades implementadas a partir do quinto nível como, por exemplo, um programa de usuário, um pacote aplicativo, ou mesmo uma pessoa utilizando um terminal. Essas atividades serão aqui designadas genericamente por processos de aplicação (PA).

Os PA solicitam do sistema serviços especilizados de alto nível tais como a comunicação terminal a terminal, correio eletrônico e, no nosso caso, um serviço de gerenciamento de arquivos (SGA).

Os PA (localizados nos equipamentos usuários da rede local) e o SGA (localizado no SA) são portanto usuários do subsistema de transporte conforme indica a figura II.5. Um PA comunica-se com o SGA através de um protocolo de alto nível denominado protocolo de gerenciamento de arquivos (PGA).

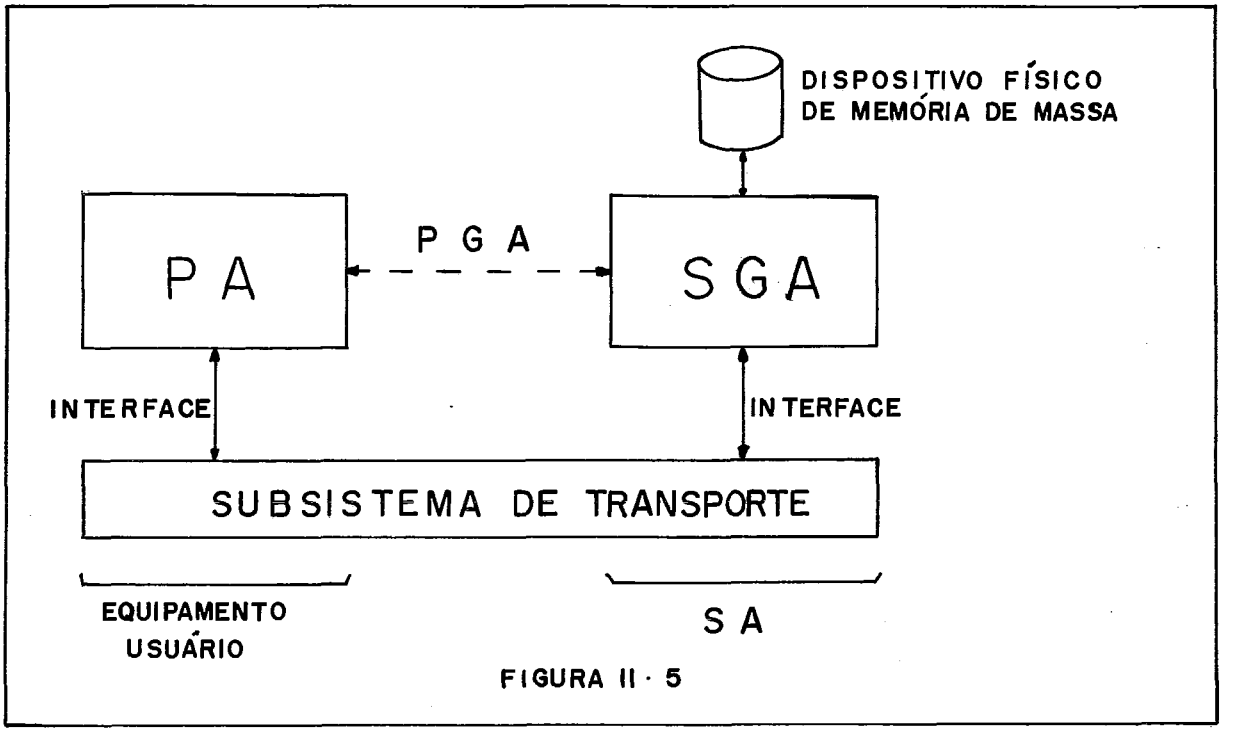


FIGURA II · 5

O SA é portanto um equipamento cujo software implementa, além do subsistema de transporte, uma camada de mais alto nível (SGA) dedicada ao gerenciamento do recurso memória de massa.

2.2 A arquitetura adotada pelo SA

Como vimos o SA é composto de dois módulos básicos: o subsistema de transporte e o SGA.

O SGA utiliza os serviços fornecidos pelo subsistema e presta serviços aos PA através do PGA. A estrutura do SGA é o tema central deste trabalho e será portanto detalhada ao longo do mesmo.

O subsistema de transporte adotado será o da Rede Local CETUS, produto já disponível no mercado nacional (vide Anexo I). Passaremos aqui a uma breve descrição dos mecanismos de comunicação oferecidos por este subsistema ao SGA.

O subsistema de transporte em questão é constituído de três processos básicos dedicados respectivamente a: recepção de mensagens da rede, transmissão de mensagens para a rede e gerenciamento de circuitos virtuais (GCV). A figura II.6 mostra a estrutura do conjunto.

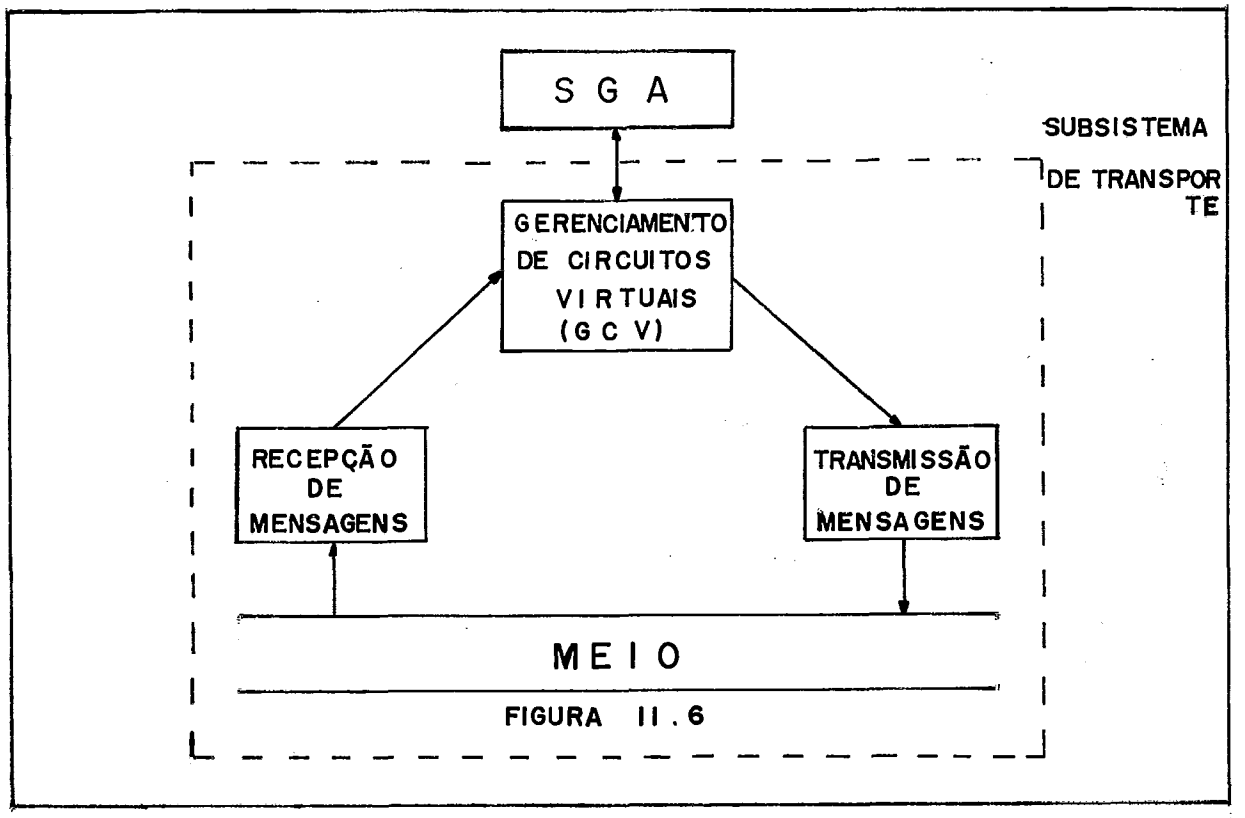


FIGURA II.6

Entre os dois tipos básicos de mecanismos fornecidos por um subsistema de transporte: circuito virtual e datagrama |Wecker, 1979| a rede CETUS optou pelo primeiro.

No primeiro caso, dados são transferidos depois que um canal lógico, denominado circuito virtual, é estabelecido entre os dois usuários do subsistema, sendo o canal desfeito ao término da comunicação. Neste caso, o subsistema se responsabiliza ainda pela ordenação das mensagens, endereçamento, controle de

erros e controle de fluxo dos dados transferidos. Essas tarefas são realizadas pelo GCV.

No segundo caso, os usuários do subsistema transmitem blocos de informação endereçados sem necessidade de estabelecer canais lógicos prévios. O serviço é não-garantido e não-sequencial e pode ser comparado ao serviço postal. Cada bloco de informação, denominado datagrama, é análogo a uma carta, completamente independente de outras cartas (datagramas) enviadas através do sistema de comunicação. As funções de reorganização e controle são deixadas para o usuário.

Ambos os esquemas apresentam deficiências relativas. O datagrama tem sobre o circuito virtual a desvantagem de sobrecarregar as tarefas do usuário; o circuito virtual por sua vez causa um "overhead" pelas mensagens de controle de abertura e fechamento de diálogo.

A alocação do recurso SA, através de um único circuito virtual, pode conduzir a implicações perigosas do ponto de vista da performance do sistema. A alocação do canal virtual por um longo tempo de conexão pode prejudicar sensivelmente o desempenho do sistema. O gerenciamento de diversos circuitos virtuais mantidos paralelamente permite que o recurso memória de massa vá sendo alocado dinâmicamente pelos vários canais à medida que operações vão sendo requisitadas pelos mesmos. Aumenta-se assim a taxa de utilização do recurso físico pelos usuários.

A rigor, o GCV seleciona os "datagramas" enviados pela comunicade de usuários e redireciona-os para os canais lógicos correspondentes, realizando os controles de fluxo e erros adequados.

2.3- A Cooperação Entre os Componentes de um Sistema Distribuído

Passaremos a analisar a cooperação num sistema distribuído integrado por uma rede local, do qual participam componentes heterogêneos. Isto será feito com base no perfil típico do usuário atual e da situação real na qual se encontra o desenvolvimento do software para redes locais.

Quando dois usuários de uma rede local realizam, através de um servidor, o compartilhamento físico e lógico de informações, estabelecem dois tipos de cooperação: a cooperação direta entre cada par usuário-servidor e a cooperação indireta entre usuário-usuário . A figura II.7 exprime melhor a idéia.

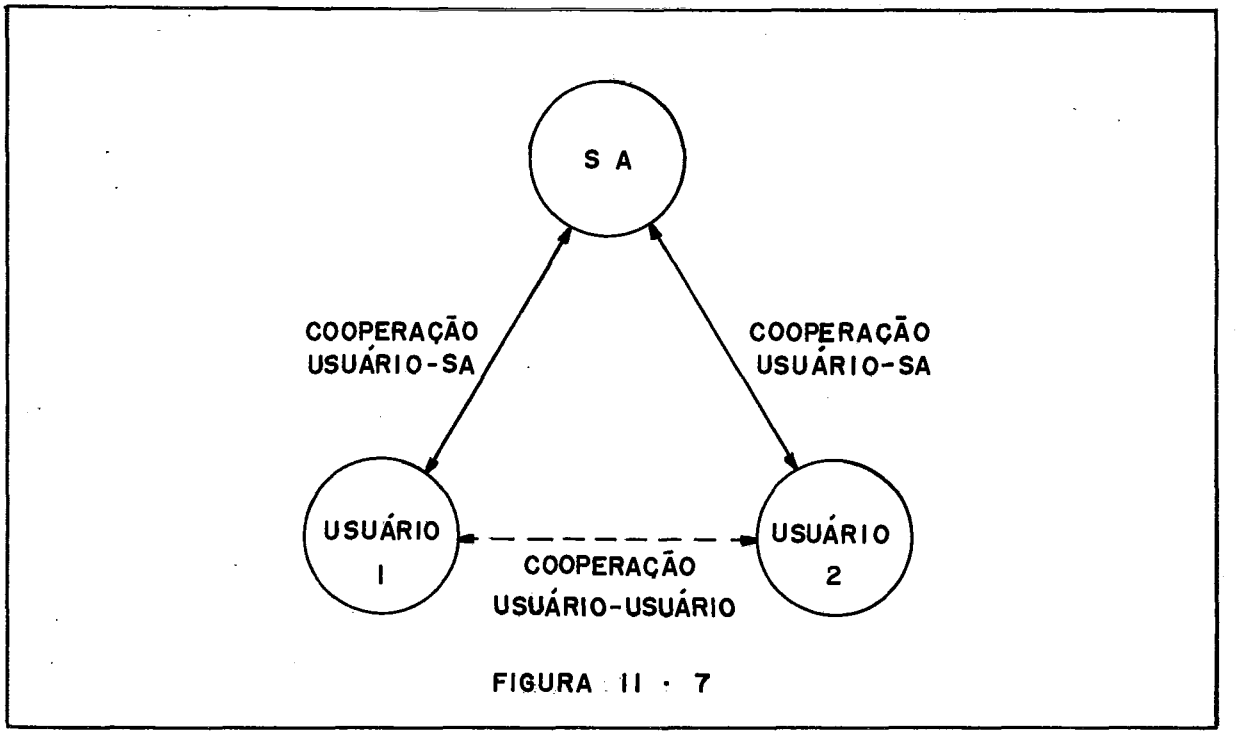


FIGURA II · 7

Mas a cooperação entre dois componentes de um sistema só é possível se a interface entre eles é compatível. Existem dois tipos de compatibilidade: funcional e protocolar

|Bochmann, 1983|.

A compatibilidade funcional existe quando um dos componentes é potencialmente capaz de executar as tarefas que o outro solicita. Caso isso não aconteça a cooperação entre eles se torna obviamente impossível.

Embora exista compatibilidade funcional, pode acontecer que os dois elementos não utilizem o mesmo conjunto de primitivas. Neste caso dizemos que não há compatibilidade protocolar. Este fato pode ser contornado introduzindo-se convenientemente um módulo de adaptação entre os processos de comunicação nos diferentes componentes.

2.3.1- Compatibilidade Usuário-Servidor

O mercado internacional de computadores exibe uma grande quantidade de produtos de marcas, portes e modelos diferentes, geralmente incompatíveis tanto em termos de hardware como de software. As memórias de massa, por exemplo, que se apresentam na prática usam diferentes mídias, diferentes formatos de gravação e estruturas de arquivos incompatíveis.

Os sistemas operacionais residentes nestas máquinas, sejam mono-usuários, concorrentes ou multi-usuários, são em geral centralizados e incompatíveis entre si. Mesmo tentativas significativas como a da Digital Research, com o Control Program For Microcomputers (CP/M), no sentido de comercializar um sistema operacional independente do hardware, apresentam "versões" incompatíveis entre si. Devido às suas características e alta portabilidade, o UNIX é freqüentemente apontado como boa opção para a padronização de software nos microcomputadores de 16 bits embora não exista um consenso sobre isto mesmo nos Estados Unidos.

A chegada das redes locais vem gerar evolução de sis-

temas centralizados para sistemas operacionais distribuídos, utilizando subsistemas de comunicação próprios para as redes utilizadas. O trabalho envolvido nesta transição deverá ser feito com base nos estudos para padronização de sistemas abertos mencionados anteriormente, evitando-se assim repetir os problemas de incompatibilidade comuns aos sistemas centralizados. Antes que um sistema operacional para redes locais seja uma realidade, os sistemas operacionais convencionais deverão operar dentro do contexto de redes locais. O usuário atravessará portanto um período de transição que suavizará a mudança para o novo ambiente. É com base nesta constatação que analisaremos a compatibilidade usuário-servidor.

Suponha que n componentes, representados por máquinas de sistemas operacionais centralizados e incompatíveis entre si, são interligados em uma rede local. Suponha, conforme discutiremos mais tarde, que o projeto do servidor ligado à mesma rede garante compatibilidade funcional aos referidos equipamentos.

Cada sistema, antes centralizado, deverá agora ganhar capacidade de acessar remotamente recursos existentes em outros pontos da rede, como por exemplo no SA. Sendo os sistemas incompatíveis entre si, é natural que não exista compatibilidade protocolar entre cada usuário e o SA, exigindo-se assim a criação de diferentes módulos de adaptação nos diferentes equipamentos.

O ideal seria que, ao implantar o módulo de adaptação no sistema operacional nativo da máquina, programas já existentes continuassem sendo usados sem alteração nos seus fontes; conseqüentemente programas que não dispõem de fonte poderiam continuar a ser utilizados. Minimiza-se assim todo o suporte de software a ser desenvolvido para o novo contexto, isto é, compi

ladores, editores, utilitários de manipulação de arquivos, etc. Trabalhos deste nível já foram realizados sobre sistemas operacionais comercialmente importantes, como o UNIX |ROWE, 1982| e o CP/M |DIGITAL, 1982|, demonstrando que, a partir de um bom trabalho de integração, um ambiente de rede local pode ser suportado por um sistema operacional já existente.

Este trabalho de implantação pode ser realizado de maneiras mais ou menos superficiais no sistema operacional em questão dependendo do grau de facilidade que o mesmo oferece a "cirurgias" e do objetivo do usuário. Usuários de alto grau de responsabilidade, como já definimos anteriormente, administram suas regiões de memória através dos próprios sistemas locais utilizando basicamente primitivas de leitura e escrita em registros. Neste caso, o SA é de mínima responsabilidade, desconhecendo a estrutura interna das informações do usuário. Este é o caso de implantação mais superficial pois basta que o "drivers" de leitura e escrita em disco local sejam substituídos por acessos remotos.

Usuários de menor nível de responsabilidade implantam um conjunto mais amplo de primitivas, tais como abertura e fechamento de arquivos, no seu sistema nativo deixando ao SA tarefas de maior responsabilidade na execução das funções correspondentes. Este nível de implantação é bem mais profundo e exige um conhecimento maior do sistema residente.

Quando, por algum motivo, torna-se impossível a implantação direta no sistema operacional nativo, o usuário só poderá acessar os recursos remotos através de programas aplicativos. Consequentemente, um programa que não dispõe de fonte exigirá que seus arquivos de trabalho sejam transportados para uma mídia local antes de ser executado. Esta característica priva o usuário de uma vantagem essencial do servidor na rede local:

o compartilhamento lógico e dinâmico de informações entre os usuários.

A figura II.8 ilustra as opções de implatanção do módulo de adaptação numa máquina genérica.

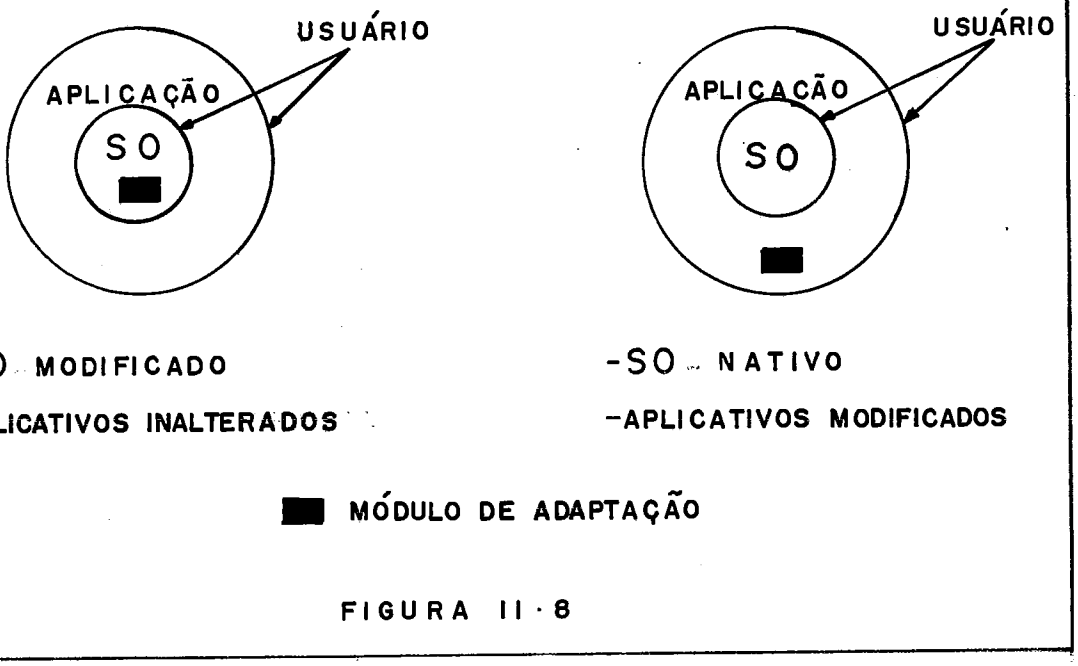


FIGURA II·8

Concluindo, é fundamental que o SA assegure a compatibilidade funcional para todos (ou quase todos) os sistemas operacionais disponíveis. Isso significa que o SA fornece um conjunto de serviços tais que o usuário consiga exprimir cada função realizada pelo seu sistema de arquivos em uma ou mais primitivas do SA. A compatibilidade protocolar fica a cargo do módulo de adaptação de cada sistema que será tão mais simples quanto mais coincidentes forem as primitivas de acesso aos serviços.

2.3.2- A Compatibilidade Usuário-Usuário

Nosso interesse é analisar o caso em que um usuário resolve acessar um arquivo criado por um sistema diferente do seu. Em outras palavras sistemas diferentes desejam fazer o

compartilhamento lógico de informações. O usuário interessado em acessar o arquivo deverá então conhecer o formato sob o qual a informação foi gravada. Normalmente, devido à incompatibilidade de formatos, o acesso ao objeto deverá ser feito via programas aplicativos para conversão de formato. Nesse caso, o SA agiu como elemento intermediário emprestando à cooperação usuário-usuário compatibilidades funcional e protocolar.

CAPÍTULO III

ESTRUTURA BÁSICA DO SISTEMA

De acordo com o modelo proposto no capítulo anterior (figura II.5) nosso problema se reduz à definição de dois elementos conceituais: serviços e protocolo.

Os serviços oferecidos pelo SGA envolvem: a organização estrutural da memória de massa e um conjunto de operações que atuarão sobre a referida estrutura. Os serviços consistem, entre outros, de: organização multi-nível para os arquivos, administração do espaço disponível, privacidade da informação armazenada, recuperação automática em caso de falhas e manipulação de objetos.

Os usuários acessam os referidos serviços através de um conjunto de primitivas que constituem o protocolo de gerenciamento de arquivos (PGA). Como exemplo, poderemos citar a criação, abertura, fechamento, leitura e escrita de arquivos.

3.1- O Aspecto Estrutural da Memória de Massa

Os conceitos apresentados ao longo deste item foram também tratados em diversos trabalhos relacionados com sistemas de arquivos |TSICHRITZIS, 1974; DONOVAN, 1972; WATSON, 1970; ORGANICK, 1972; THOMPSON, 1974; THOMPSON, 1978|.

3.1.1- Apresentação da Estrutura

O conteúdo da memória de massa inclui dados, programas, diretórios, apontadores, controles, descritores, enfim todo o tipo de informação, redundante ou não, que necessite ser preservada.

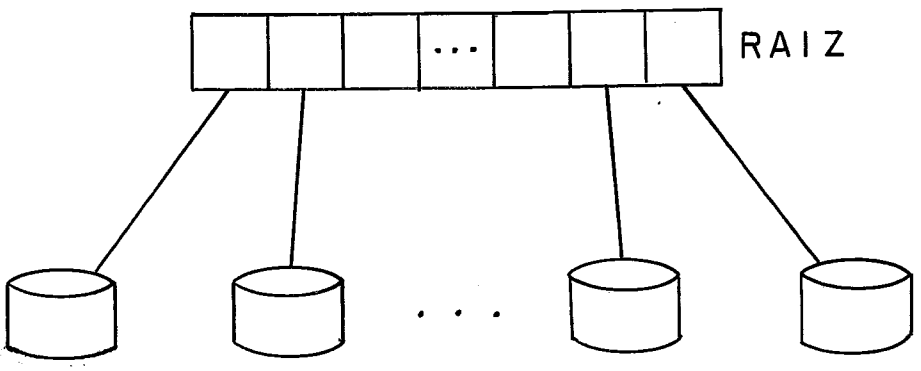
Estas informações estão distribuídas em entidades lógicas, denominadas objetos, que se articulam dentro de uma organização estrutural. Cada objeto, dentro desta organização, é suportado por uma estrutura de dados própria. É sobre os objetos que operam as primitivas de manipulação lógica da memória secundária. Arquivos, diretórios e registros são exemplos clássicos de objetos, normalmente disponíveis em qualquer sistema de arquivos convencional. Serão portanto aqui adotados dentro de uma estrutura suficientemente versátil para facilitar a correspondência funcional mencionada anteriormente.

Um arquivo é uma sequência de informações. Associado a ele existe um descritor que guarda: nome, atributos e direitos de acesso. Existem arquivos especiais, denominados diretórios, que concentram informações sobre o tipo e a localização de outros arquivos. Cada elemento de um diretório (entrada) é portanto um descritor de outro arquivo.

O SA associa a cada usuário uma região de memória, denominada minidisco, com tamanho máximo fixado no momento da sua criação. A descrição dos minidiscos é mantida pelo SA num diretório central, denominado RAIZ, conforme mostra a figura III.1.

Cada vez que é solicitada a criação de um objeto minidisco, o SA cria a "raiz do minidisco" ou o seu "diretório - raiz" que será o diretório de mais alto nível daquele usuário.

O dono do minidisco poderá então criar entradas no seu diretório-raiz para arquivos próprios. Fica assim esboçada a estrutura de um sistema de arquivos que constitui uma árvore de dois níveis conforme mostra a figura III.2. Os diretórios são representados por retângulos e os arquivos não-diretórios por círculos.



MINIDISCOS

FIGURA III·1

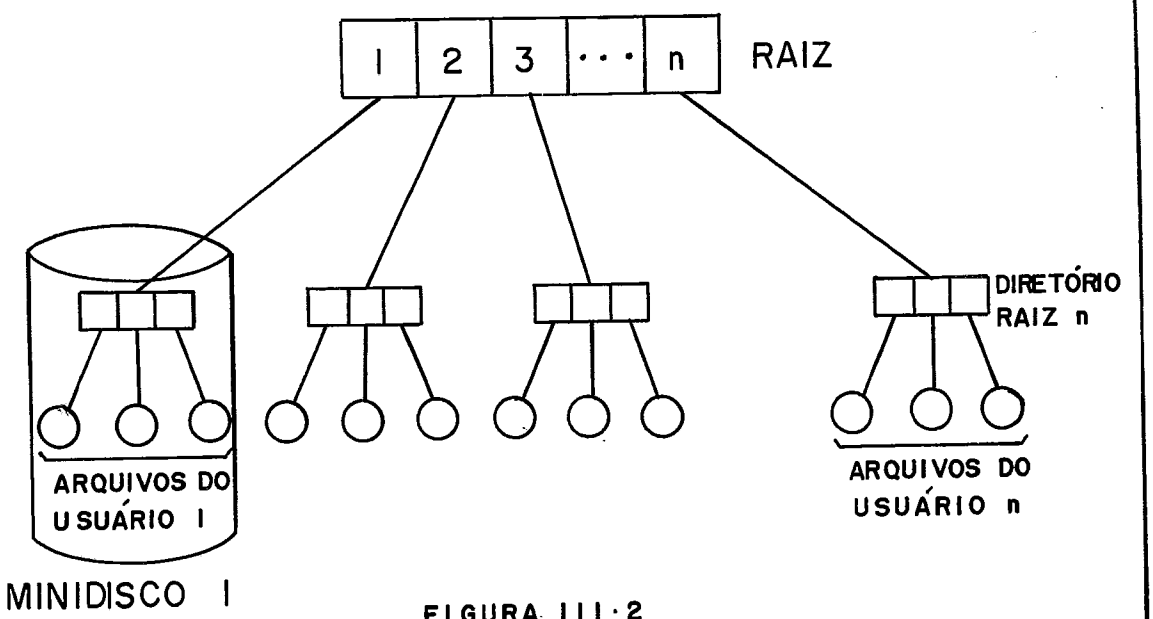


FIGURA III·2

A estrutura em dois níveis pode ser generalizada permitindo que a raiz do minidisco aponte para outros diretórios e assim sucessivamente, conforme a figura III.3.

Confere-se assim ao usuário maior liberdade para estruturar seus arquivos na forma apropriada para o seu problema; arquivos podem então ser agrupados por projetos e subprojetos (por exemplo um subdiretório com rotinas matemáticas que por sua vez geraria subdiretórios respectivamente para rotinas estatísticas, matriciais e de resolução de equações). A estrutura dos sistemas operacionais convencionais são casos particulares desta organização multi-nível, sendo que alguns, como o UNIX |THOMPSON, 1974; 1978|, a utilizam na sua forma mais geral.

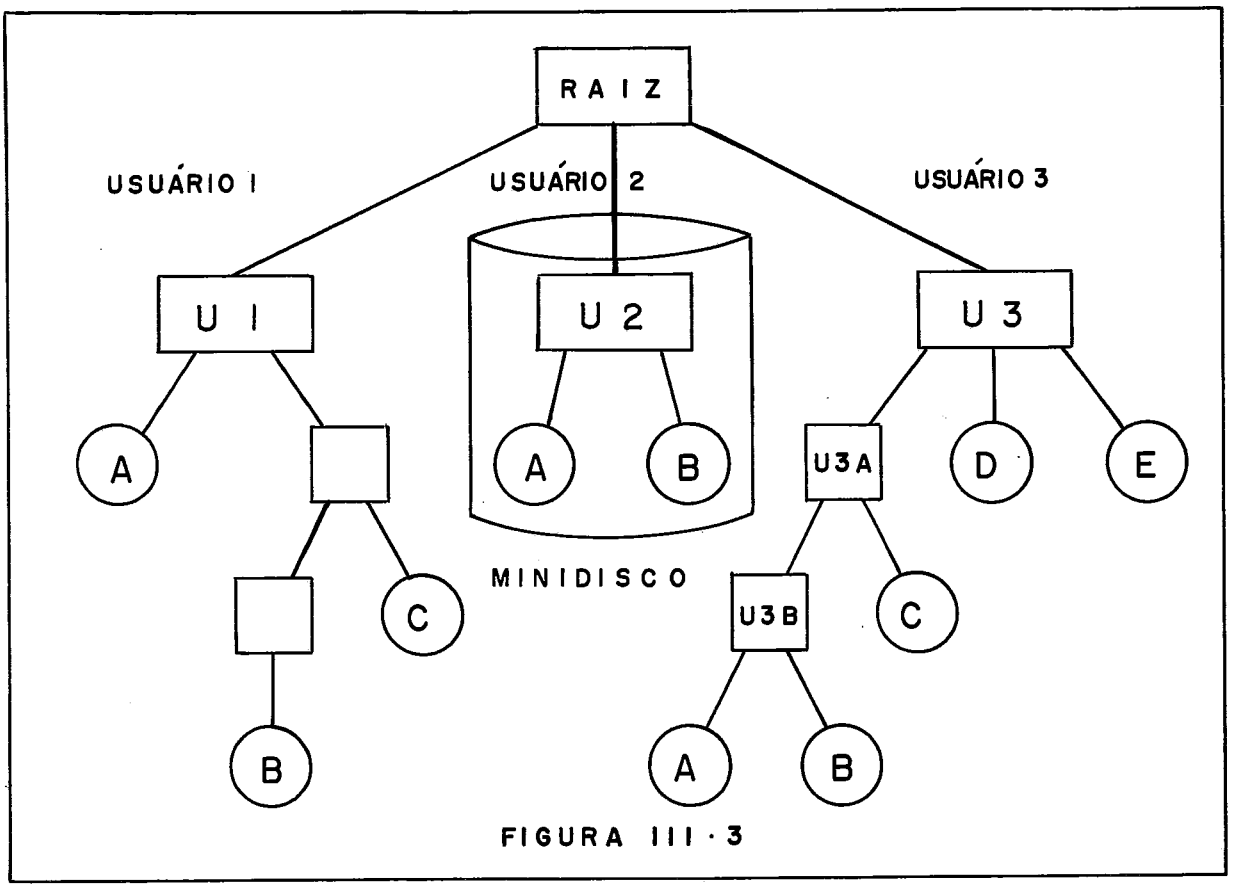


FIGURA III·3

3.1.2- Identificação dos Arquivos no SA

No momento em que uma entrada é criada em um diretório, são associados a ela, entre outras informações, um número denominado código de identificação do arquivo (CIA) e uma ca-

deia alfanumérica denominada nome de referência.

O CIA é suficiente para localizar o arquivo que a referida entrada descreve. Normalmente, costuma ser mais eficiente referenciar um objeto pelo seu CIA.

O nome de referência, por sua vez, é apenas a parte terminal do nome que identifica o objeto. O nome completo de um arquivo ou diretório é o resultado de uma concatenação dos nomes de referência, desde a RAIZ até o diretório que contém a referida entrada. Por exemplo, utilizando-se o delimitador "/", na figura III.3, o nome completo do arquivo A pertencente ao diretório do usuário 1 é RAIZ/U1/A ou /U1/A pois, normalmente, se omite o nome da RAIZ. O nome de referência do mesmo arquivo é simplesmente A. Em um mesmo diretório duas entradas não podem incorporar o mesmo nome de referência. No entanto, nomes de referência podem coincidir em dois (ou mais) diretórios distintos.

Adotaremos também uma convenção bastante comum que ajuda a encurtar a identificação de um arquivo: o "diretório corrente" |Organick, 1972|. Quando o nome de um arquivo se inicia com um caracter diferente de "/", considera-se que ele está automaticamente relativo ao diretório corrente. Assim, no exemplo da figura III.3, se o diretório corrente para o usuário 3 for /U3/U3A, o arquivo B poderá ser referenciado simplesmente pelo nome U3B/B.

3.1.3- A estrutura Interna de um Arquivo

Um dos requerimentos de um sistema de gerenciamento de recurso físico é tornar o hardware do sistema transparente ao usuário.

O dispositivo de memória de massa é um vetor de unida

des físicas diretamente acessíveis denominadas registros físi - cos.

Um bloco é um conjunto de registros físicos contí - guos.

Para o sistema, a unidade de manipulação da memória de massa é o bloco; otimizam-se assim os tempos envolvidos nos acessos ao dispositivo. O sistema define portanto o tamanho do bloco que mais lhe convier dependendo das características físicas da memória e do perfil das operações que se realizam sobre a mesma.

O usuário vê o arquivo como um vetor ordenado de registros lógicos ignorando a situação dos mesmos em relação aos blocos. Devido à heterogeneidade dos usuários é nossa proposta permitir que o tamanho do registro lógico seja fixado no momento da criação de cada arquivo; minimizaremos assim uma série de conversões na fase de adaptação protocolar.

O diretório é um caso particular de arquivo que só pode ser alterado pelo sistema. Num diretório, cada registro lógico (entrada) abriga um conjunto de informações detalhadas sobre outro arquivo (diretório ou não), denominado descritor.

São os seguintes os tipos de informações existentes no descritor:

. o nome de referência do arquivo

. o CIA do arquivo

. o dono do arquivo

. o tipo do arquivo

. os direitos de acesso sobre o arquivo

. o tamanho do arquivo

. o tamanho do registro lógico

. a localização física dos registros lógicos (mapa do arquivo)

É aconselhável que o tamanho da entrada do diretório seja o menor possível para minimizar acessos a disco durante as buscas. Isolaremos assim no diretório apenas os dois primeiros itens (nome e CIA). O restante da informação será colocado numa tabela, denominada ÍNDICE, localizada numa região determinada do dispositivo. O próprio CIA indicará a entrada do ÍNDICE correspondente àquele arquivo. O acesso adicional criado em função da estrutura ÍNDICE torna-se assim direto e rápido. Uma entrada genérica da tabela ÍNDICE será aqui denominada nodo.

3.1.4- Alocação de Espaço Físico Para um Arquivo

A unidade física de alocação de espaço é o bloco. Quando um usuário acessa um registro lógico, o bloco que o contém é copiado para a memória principal. As operações de leitura subseqüentes em torno daquele registro lógico poderão então aproveitar os dados já existentes em memória principal, minimizando assim os acessos a disco.

Os n primeiros blocos alocados para um arquivo estão incluídos no nodo correspondente da tabela ÍNDICE. Se o arquivo for maior que n blocos, um campo adicional no nodo (Próximo1) apontará para um bloco que contém os endereços dos próximos m blocos pertencentes ao arquivo. Se n + m blocos não forem suficientes, um campo adicional no nodo (Próximo2) referenciará um bloco que aponta para m blocos, cada um contendo ainda m blocos pertencentes ao arquivo. Um arquivo poderá ter portanto, no máximo, $n + m + m^2$ blocos, conforme mostra a figura III.4.

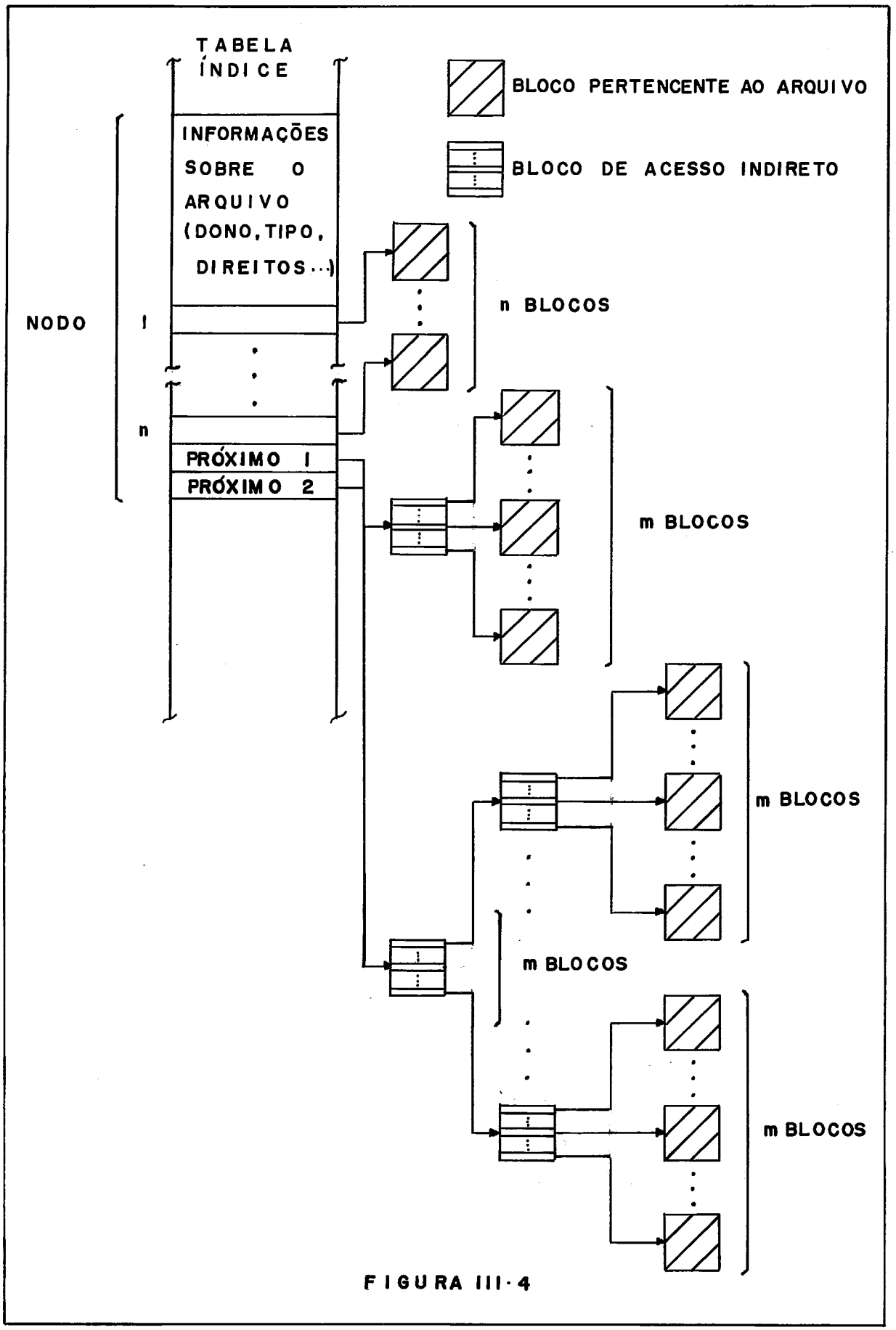


TABELA
ÍNDICE
INFORMAÇÕES
SOBRE O
ARQUIVO
(DONO, TIPO,
DIREITOS...)
NODO
1
n
PRÓXIMO 1
PRÓXIMO 2
BLOCO PERTENCENTE AO ARQUIVO
BLOCO DE ACESSO INDIRETO
n BLOCOS
m BLOCOS
m BLOCOS
m BLOCOS
m BLOCOS

FIGURA III-4

Esta organização otimiza o tempo de acesso para arquivos com tamanho menor ou igual a n blocos. Como o conteúdo do nodo correspondente a um arquivo é copiado para a memória principal enquanto o mesmo permanece aberto, evitaremos acessos indiretos a disco se este arquivo tem, no máximo, n blocos. Caso o arquivo tenha mais de n blocos, a organização procura minimizar os acessos na localização de um determinado registro lógico.

3.1.5- Administração do Espaço Livre

A escolha da política de administração do espaço livre disponível é um compromisso entre a quantidade de memória principal utilizada neste controle e o número de acessos a disco envolvidos. No caso ideal toda a informação a respeito do estado de alocação dos blocos estaria disponível em memória viva; assim todas as solicitações de alocação teriam uma resposta praticamente imediata. Na prática, no entanto,os dispositivos de memória de massa têm sua densidade aumentada dia a dia e o software de gerenciamento não pode ficar limitado pela capacidade de memória principal do equipamento. Devem-se portanto prever esquemas que minimizem os acessos utilizando uma fração máxima da memória principal.

O controle do espaço disponível é realizado através de uma tabela de alocação de blocos, armazenada em memória secundária. Nesta tabela cada bit representa o estado de um bloco (ocupado ou livre). Dada a posição de um determinado bit, um cálculo simples indica o bloco físico ao qual corresponde.

Parte da tabela é sempre mantida em memória principal; com um único acesso, ela fornece portanto diversos blocos livres. Quando eles se esgotam, a memória principal é recarregada com uma partição diferente da tabela.

Devido à forma compactada com que cada bloco é repre-

sentado (1 bit), o gasto de memória de massa para este controle é desprezível (cerca de 0,02% da memória total para blocos de 512 bytes).

3.1.6- "Overhead" de Memória Imposto Pela Organização Estrutural.

A memória de massa está dividida em cinco regiões: a primeira inclui um bloco com parâmetros de controle tais como o tamanho total da memória e das demais regiões; a segunda contém o mapa de alocação de blocos; a terceira está reservada para as listas de intenções (a serem descritas adiante); a quarta corresponde à tabela ÍNDICE; e a quinta, incluindo o restante do disco, abriga todos os blocos a serem alocados dinamicamente para arquivos, diretórios e blocos de acesso indireto aos arquivos. A figura III.5 mostra a estrutura descrita.

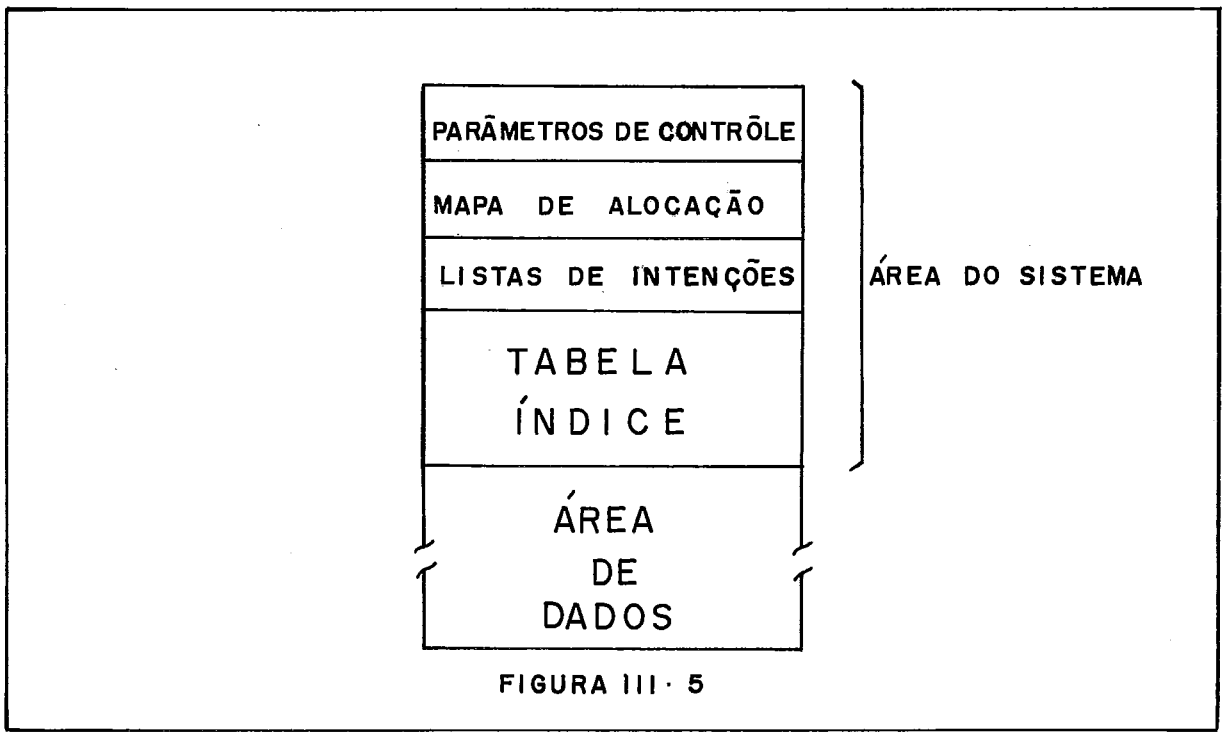


FIGURA III.5

As quatro primeiras regiões são totalmente ocupadas por informações estruturais e, portanto, constituem um "overhead" de espaço obrigatório sobre a memória de massa. Na

última região, o espaço efetivo alocado para informação estrutural varia dinamicamente não sendo portanto determinado.

Dentre as quatro primeiras regiões, aquela ocupada pela tabela ÍNDICE é a mais relevante pois é proporcional ao número de arquivos (diretórios ou não) que o sistema suporta.

Na nossa implementação adotamos blocos de 512 bytes por ser uma opção mais atraente, do ponto de vista de velocidade de transferência, para os controladores de disco rígido utilizados. Consideramos ainda que um apontador de 3 bytes consegue referenciar até 16M blocos o que corresponde a uma disco da ordem de gigabytes. Daí deduzirmos que um nodo de 64 bytes consegue referenciar por si só 15 blocos que constituem um arquivo de 7,5K bytes para o qual os acessos não dependem de localização indireta de registros lógicos.

O número de nodos da tabela ÍNDICE pode ser dimensionado no momento da geração do sistema. Para um caso particular de cerca de 400 arquivos para cada 1M bytes de memória o "overhead" fixo é da ordem de 2% e, portanto, irrelevante.

Suponha ainda como exemplo que U usuários que antes trabalhavam em computadores monousuário utilizando o CP/M |DIGITAL, 1979| agora interligam suas máquinas através de uma rede local. Suponha que eles resolvem transferir os seus arquivos para o SA, mantendo a estrutura anterior (Figura III.6) na qual cada um deles manipulava 16 disquetes de 250K bytes com cerca de 100 arquivos cada um.

Cada diretório-raiz (DRi) terá portanto 16 entradas cada uma delas definindo um outro diretório com cerca de 100 entradas. Neste caso, cada diretório-raiz gastaria 1 bloco e os demais 4 blocos cada, totalizando 65U blocos (U + 4 x 16 x U) .

Mantendo a quantidade de memória utilizada para dados por cada disquete de usuário (cerca de 240K bytes), os diretórios ocupariam (65U/2)K bytes/(65U/2 + 240.16U)K bytes sendo portanto inferior a 1% o "overhead" associado.

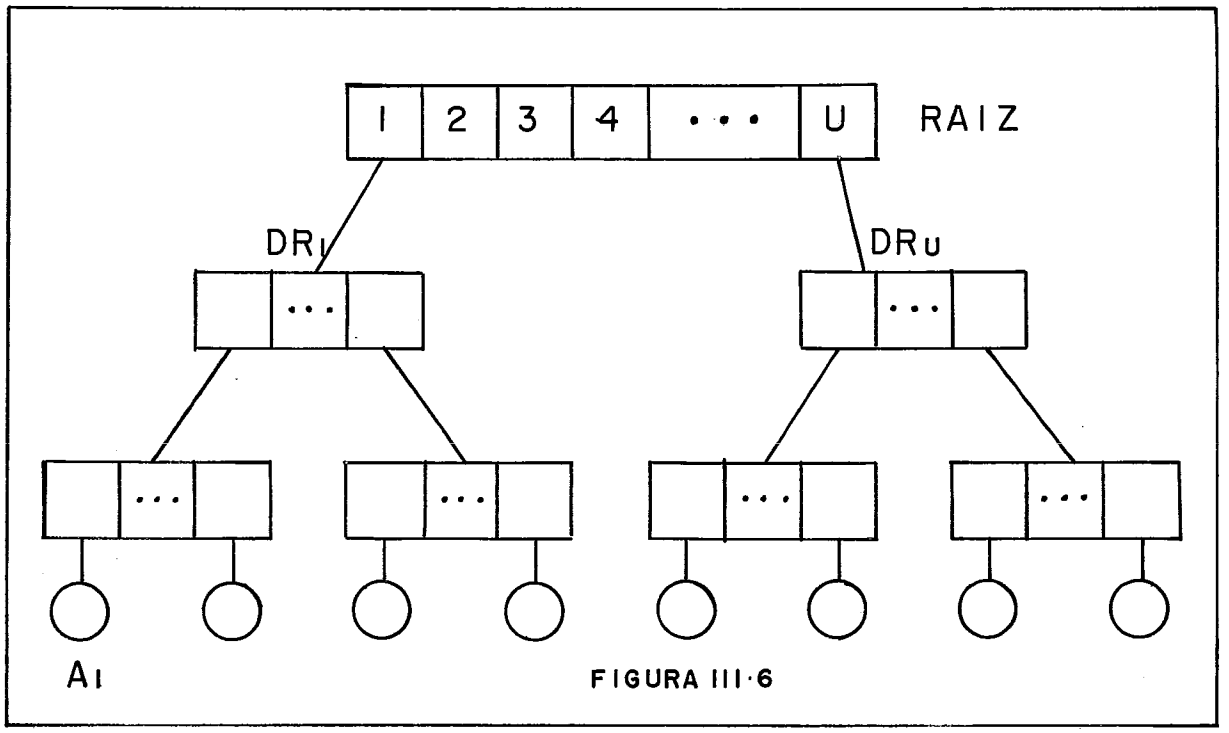


FIGURA III·6

3.2- Manipulação Lógica dos Objetos

O objetivo deste item é descrever os principais mecanismos de manipulação da estrutura adotada, fornecendo alguns esclarecimentos sobre aspectos de implementação.

3.2.1- O Gerenciamento da Memória de Massa

Quando o sistema cria um minidisco, introduzindo um novo usuário na raiz do SA, recebe o número máximo de blocos que o minidisco poderá ocupar. A alocação efetiva de blocos para esse usuário, no entanto, vai-se fazendo à medida que a mesma é solicitada por primitivas de criação de arquivos ou escrita em registros.

O controle da alocação de espaço é realizado com base no mapa de alocação que é mantido em memória de massa. Uma cópia de parte desta tabela fica em memória principal sendo periodicamente atualizada em memória secundária.

3.2.2- Como Refenciar um Arquivo

Um arquivo fica perfeitamente definido pelo CIA ou pelo seu nome completo. No entanto é possível que os usuários, por comodidade, façam referências ambíguas que o sistema deve resolver a partir de uma convenção.

Em grande parte dos casos,apenas o nome de referência do arquivo é utilizado. Em outras palavras, dá-se o nome do arquivo destituído do nome completo do diretório que o defi-

ne ou diretório ancestral.

Há sistemas que realizam então uma busca num vetor ordenado de diretórios procurando por uma entrada que coincida com o nome de referência fornecido. No nosso caso, realizaremos a busca apenas sobre o "diretório corrente". Quando uma conexão lógica é estabelecida com o usuário, o seu diretório-raiz é automaticamente escolhido como corrente. Através de primitiva adequada, o usuário poderá alterá-lo.

Para algumas primitivas, como a de remoção de arquivos, a cadeia que define um nome de referência pode conter em parte, ou totalmente, caracteres de indeterminação para os quais qualquer valor é considerado coincidente no momento da comparação.

3.2.3- Proteção da Informação

O compartilhamento de arquivos entre comunidades de usuários cria a necessidade de garantir a privacidade e a integridade da informação armazenada. À medida que facilidades de acesso são oferecidas a terminais anônimos e remotos, uma grande responsabilidade é entregue ao sistema centralizador dos dados.

São as seguintes as facilidades comumente exigidas que aqui adotaremos:

- proteção contra um usuário que tenta se passar por um outro. Neste caso, o sistema exige a apresentação de uma password no momento da abertura da sessão |WATSON, 1970; GAIT; 1978|.
- proteção contra o acesso ao arquivo por parte de um usuário que não está autorizado a fazê-lo. Neste caso, o SA associa ao objeto direitos de acesso que

são armazenados como atributos no seu descritor |WATSON, 1970; ROBERTSON, 1980|.

- proteção contra erros gerados pelo próprio dono do arquivo que, neste caso, poderá usar os direitos de acesso acima para se proteger dos seus próprios enganos. Mecanismos de "backup" também poderão proteger o dono do arquivo de perdas acidentais.

- proteção contra o acesso concorrente a um objeto , gerando um estado inconsistente para o mesmo. Para este caso, o SA define uma política de controles de acesso |STURGIS, 1980; DIGITAL, 1982|.

- proteção contra falhas no sistema, assegurada por mecanismos de recuperação automática |DION, 1980; STURGIS, 1980; MITCHELL, 1981|.

3.2.4- Os Direitos de Acesso

Usuários podem ter os seguintes direitos de acesso sobre um arquivo: nenhum acesso, acesso apenas para leitura ou acesso para leitura e escrita. Diferentes usuários podem receber diferentes direitos de acesso para um determinado objeto . No formato mais geral, deveria ser mantida uma lista de usuários com seus respectivos direitos de acesso. Simplificaremos aqui definindo duas classes de usuário: o usuário que criou o objeto e os demais usuários. No momento da criação do arquivo, o direito de acesso do usuário, denominado direito de acesso próprio, é automaticamente fixado para o modo leitura e escrita. O direito de acesso para os demais usuários, denominado direito de acesso público, é fixado pelos atributos passados como parâmetros. Apenas o usuário que criou o arquivo tem direito de alterar esses atributos.

3.2.5- Os Controles de Acesso

Além dos direitos de acesso que ajudam a assegurar a privacidade dos arquivos, o SA tem que fornecer mecanismos de controle (dinâmico) de acesso que assegurem a consistência da informação armazenada.

Quando o acesso concorrente a um objeto é permitido, o SA realiza um bloqueio seletivo para evitar que dois usuários alterem o conteúdo simultaneamente causando resultados desastrosos. O bloqueio atua então como um mecanismo de sincronização que pode ser acionado implicitamente, por ocasião da abertura de um arquivo, ou explicitamente através de primitivas específicas.

Ao abrir um arquivo, um usuário pode escolher um dentre três modos de operação:

- Exclusivo

  Um usuário pode abrir um arquivo neste modo apenas se ele não está aberto naquele momento; é o modo mais convencional.

- Leitura

  Um usuário pode abrir um arquivo neste modo se o arquivo não está aberto naquele momento ou se ele está aberto em modo leitura. Assim diversos usuários podem acessar o arquivo simultaneamente, sendo permitidas apenas operações de leitura.

- Compartilhado

  Um usuário pode abrir um arquivo neste modo se o arquivo não está aberto naquele momento ou se ele está aberto em modo compartilhado. Este modo permite que diversos usuários acessem paralelamente o mesmo arquivo para leitura e escrita. Se utilizado em combinação com primitivas adequadas de bloqueio e desbloqueio (explícitos) de registros, este modo po

derá aumentar o paralelismo de operação em muitas aplicações nas quais usuários desejam alterar o mesmo arquivo em posições mutuamente exclusivas.

3.2.6- Manutenção da Integridade do Dados

i) Os Conceitos de "Transaction" e Atomicidade

O SA não tem capacidade de interferir sobre a lógica de manipulação dos dados do usuário mas tem o dever de protegê-los contra inconsistências geradas por eventos que fogem do controle dos usuários tais como acesso concorrente sobre arquivos, falhas de comunicação e falhas no hardware do SA.

Para sincronizar a concorrência no compartilhamento de dados, foram adotados controles de acesso descritos anteriormente. Para salvaguardar a consistência diante de falhas foram introduzidos os conceitos de "transaction" e indivisibilidade de operações dos quais trataremos a seguir.

Um pedido de serviço ao SGA desencadeia uma série de operações de escrita sobre os objetos residentes em memória de massa. "Transaction" é um agrupamento destas operações constituindo operações lógicas mais amplas. A escolha das operações que serão agrupadas para formar uma "transaction" depende dos objetivos do sistema. Conforme veremos mais tarde, é possível ao usuário definir agrupamento de acordo com seus objetivos particulares.

O conceito de "transaction" ganha importância prática quando aliado ao de indivisibilidade resultando numa operação denominada "transaction atômica". Uma "transaction" é atômica quando, ou todas as operações nela envolvidas são levadas a termo, ou nenhuma delas

será realizada, independentemente da ocorrência de falhas no decorrer da sua execução. O diagrama de estados (figura III.7) permite visualizar uma "transaction" evoluindo de um estado inicial até um estado final através de estados intermediários gerados pelas operações envolvidas. Enquanto um dos estados intermediários, denominado estado crítico, não for atingido, uma falha provocará o aborto da "transaction". Uma vez atingido o estado crítico, a "transaction" chegará obrigatoriamente ao estado final. Neste caso, a ocorrência de uma falha provoca o retorno ao estado crítico e a transaction continua evoluindo a partir dali.

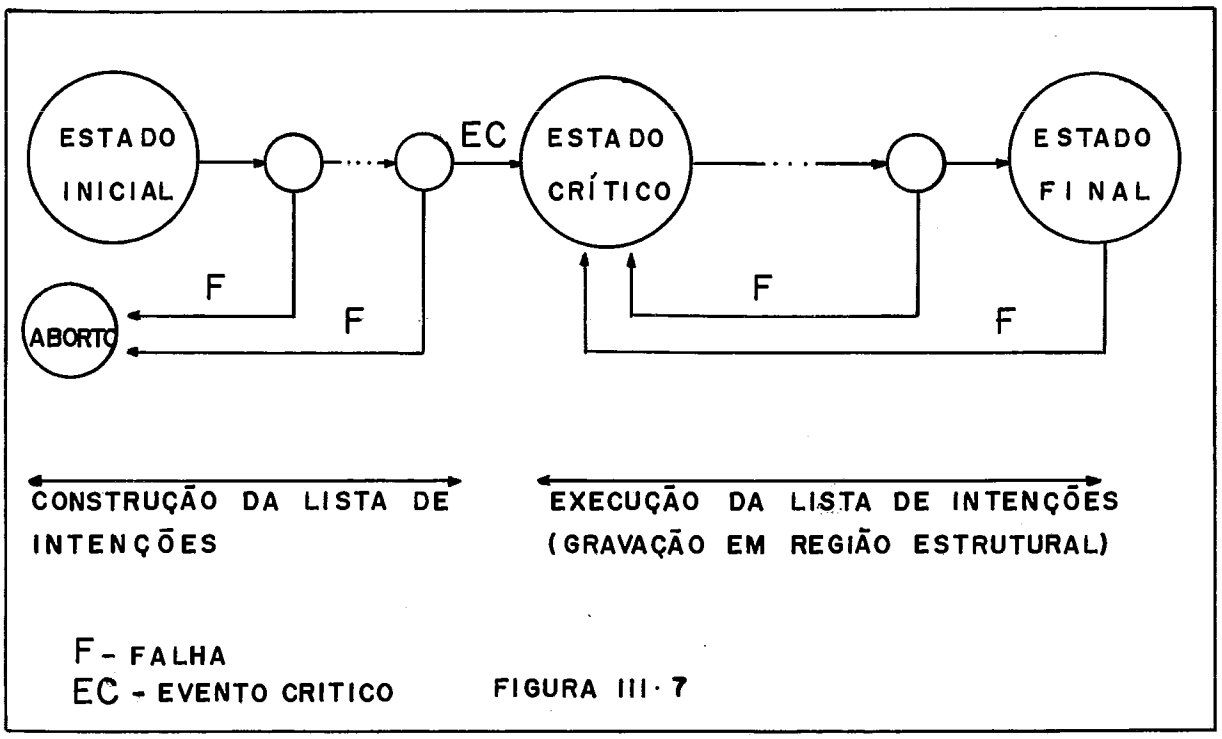


FIGURA III·7

Suponha, por exemplo, que o SGA necessite alocar blocos para um arquivo. Para tal deverá atualizar o conteúdo do nodo do arquivo assim como a tabela de alocação geral da memória. Esta operação envolve portanto uma série de escritas em regiões distintas da memória de massa. Se ocorrer uma falha neste ínterim um estado inconsistente pode ser gerado. Operações que en-

volvem alterações na região estrutural do sistema são portanto casos típicos para os quais os mecanismos de atomicidade colaboram para manter a integridade dos dados. Uma "transaction" atômica será por nós designada simplesmente por "transaction".

Há sistemas que suportam primitivas explícitas de início e término de "transaction" |STURGIS, 1980|. No SA, todas as primitivas que envolvem alteração em blocos estruturais constituirão "transactions" (remoção de arquivo, alteração de atributos, fechamento de arquivos, etc.). A primitiva de escrita em registro lógico de arquivos tipo não-diretório merece uma análise especial.

Quando um usuário abre um arquivo, realiza diversas escritas sobre o mesmo e finalmente o fecha poderá ou não estar realizando uma única "transaction" sobre este objeto. Em caso positivo, o estado crítico é atingido após o fechamento do arquivo; se ocorrer uma falha entre as operações de escrita, o SA reconstituirá o arquivo ao seu estado original. Em caso negativo, o arquivo ficará com parte dos blocos alterados. O SA dá ao usuário a liberdade de escolher entre as duas opções no momento da abertura do arquivo, definindo para tal um quarto modo de abertura denominado exclusivo especial para o qual realiza-se uma "transaction". O modo exclusivo, por questão de ênfase, será chamado daqui por diante de exclusivo normal.

Ao escolher entre um dos modos, o usuário deverá avaliar o benefício obtido pela atomicidade em face do "overhead" adicionado à operação. Deve-se considerar que há arquivos para os quais tal mecanismo é desnecessário e até indesejável. Por exemplo, é mais razoável que um usuário, editando um arquivo, não perca, por questão de uma falha, toda a informação que já havia introduzido via terminal. Uma tarefa de compilação, por

outro lado, devido à facilidade de reprodução, não justifica "overhead" adicional.

ii) Deteção de falhas

O SA pode detetar uma falha em um dos seguintes casos:

- o SA deteta algum problema de hardware
- o SA deteta um mal-funcionamento do programa
- o SA deteta que o usuário envolvido na "transaction" entrou em pane; este fato é sinalizado pelo nível de transporte que acusa erro ou quebra na comunicação
- o SA deteta, através da sinalização de um "time-out", que o usuário está monopolizando o circuito virtual sem utilizá-lo efetivamente.

Ao detetar uma anormalidade, o SA inicia um processo de recuperação automática na tentativa de impedir a geração de estados inconsistentes para o sistema ou para o usuário.

O processo de recuperação pode ser ou não bem sucedido dependendo dos danos causados pela referida falha. O SA prevê portanto recuperação apenas para alguns tipos de falha; as demais são denominadas catastróficas. A escolha do nível de falha que se deseja recuperar influencia aspectos delicados no projeto pois, quanto mais graves são os danos sujeitos a recuperação, maior é o "overhead" de execução e a manutenção de estruturas adicionais. Existem falhas que: destroem o conteúdo da memória principal; destroem todo o conteúdo do disco; corrompem apenas um bloco devido a um transiente na alimentação ou a uma falha na eletrônica do disco; desviam o programa da sua execução normal. As duas últimas são menos graves e provavelmente, terão uma recuperação efetiva; para as outras um sistema de "back-up" é a única solução garantida.

iii) Implementação da Indivisilidade de uma "Transaction"

Enquanto em progresso e, antes de atingir o estado crítico, os estados intermediários da "transaction" vão sendo acumulados em memória principal numa estrutura denominada lista de intenções. Quando todas as operações lógicas envolvidas na "transaction" são determinadas, a referida lista é gravada em uma região reservada da memória de massa com uma marca de fim. Neste momento foi atingido o estado crítico e a "transaction" se torna irreversível. A partir deste momento, cada uma das intenções pertencentes à lista vai gerando de fato modificações na região estrutural do sistema; estas operações traduzem os estados que se seguem ao estado crítico. Só após a execução de todos os itens é que a marca de fim é removida. A figura anterior ilustra a construção da lista de "intenções" ao longo da "transaction". O evento crítico corresponde à ativação da marca de fim.

O algoritmo de recuperação automática procura em disco todas as listas de intenções com marca de fim ativada, levando-as a termo. Durante as tentativas de recuperação, pode ainda haver falhas. Só quando o algoritmo não encontra mais nenhuma intenção é que o sistema se encontra num estado de alocação consistente. Pode acontecer portanto que uma lista de intenções seja, em parte, executada diversas vezes para só então ser inteiramente executada e removida. Outra situação curiosa se dá quando uma falha ocorre logo depois que o SA executa uma lista mas logo antes que a marca de fim seja removida. Esta lista será portanto totalmente reexecutada sem necessidade.

O conteúdo da lista de intenções exprime uma série de alterações na região estrutural do sistema(nodos, diretórios, mapa de alocação). Quando um arquivo é aberto no modo exclusivo-especial, ficam envolvidas na "transaction" escritas em blocos de usuário envol - vendo assim um grande volume de dados na "transaction". Neste caso, a escrita do bloco é rea lizada imediatamente sobre um bloco livre e, na lis ta de intenção, fica incluída apenas a substituição do bloco antigo pelo bloco novo no mapa de alocação do arquivo. Só com o término da "transaction" é que os blocos reservados substituem efetivamente os anti- gos. Se uma falha ocorrer neste ínterim, os blocos reservados são de novo liberados e todas as opera- ções de escrita ficam sem efeito, de acordo com o de sejo do usuário.

iv) Redundância Estrutural.

Quando uma falha provoca a destruição de um bloco estrutural da memória de massa, a reexecução das lis tas de intenções não é suficiente para devolver a consistência ao sistema. Para tentar preservar seu conteúdo, utiliza-se de memória estável |STURGIS, 1980| que consiste na duplicação do bloco em questão. Os blocos que compõem cada par devem estar estrategica mente localizados no disco de modo a minimizar a possibilidade de adulteração simultânea de ambos. Diante de uma anormalidade a informação estrutural poderá ser re construída. Se um dos blocos ocasiona erro de checksum na leitura, o bloco bom é copiado sobre o ruim. Se ambos têm conteúdos válidos mas diferentes, uma falha deve ter ocorrido depois da primeira escri

ta e antes da segunda e, portanto, o conteúdo do primeiro bloco é o mais atualizado; por isso copia-se o primeiro bloco no segundo.

As listas de intenções, devido ao seu importante papel na manutenção da atomicidade das "transactions", são residentes em memória estável, assim como os blocos com informação estrutural.

v) O Procedimento do Usuário Diante das Panes do SA

O usuário deve saber conviver e tomar decisões diante do aborto de "transactions" no SA. Em geral, quando isto acontece, o usuário solicita a repetição da mesma "transaction". No entanto, as operações solicitadas podem ter sido resultado de conclusões sobre dados compartilhados que agora podem ser diferentes, exigindo repetição de "transactions" anteriores. Cabe assim ao usuário reconhecer o tipo de ocorrência e contornar adequadamente a situação.

3.3 As Primitivas de Comunicação com o SGA

3.3.1 A Escolha das Primitivas

O usuário da rede local comunica-se com o SGA através dos serviços fornecidos pelo subsistema de transporte da rede . Para tal realiza um pedido de estabelecimento de circuito virtual que, ao ser aceito, estabelece o diálogo com o elemento destino (no caso o SA) até que o usuário tome a iniciativa de encerrá-lo através de um pedido de desligamento. O protocolo de alto nível fica portanto independente da topologia específica da rede utilizada.

A escolha das primitivas que compõem o protocolo de

alto nível deve ser orientada por dois objetivos principais: viabilizar e facilitar o acesso aos serviços disponíveis no SGA.

Viabilizar o acesso equivale a garantir que todo e qualquer serviço pode ser efetivamente requisitado pelo usuário através de uma ou mais primitivas. Não haveria sentido, por exemplo, em definir "passwords" no sistema sem oferecer a primitiva que permite alterá-las.

A facilidade de acesso é tanto maior quanto maior for a correspondência entre as primitivas adotadas e as já existentes nos sistemas operacionais usuários. Esta característica facilita a compatibilidade protocolar usuário-servidor. O Anexo II mostra um exercícioprático de compatibilização protocolar entre o SA e um sistema operacional comercial: o CP/NET [DIGITAL, 1982].

Seguindo esta orientação, definiremos um conjunto de primitivas através do qual podemos acessar todos os serviços descritos neste capítulo.

- Primitivas de Controle de Sessão
  - . abertura de sessão
  - . encerramento de sessão

- Primitivas de manipulação de arquivos
  - . criação de arquivo
  - . abertura de arquivo
  - . fechamento de arquivo
  - . leitura randômica de arquivo
  - . leitura sequencial de arquivo
  - . escrita randômica de arquivo
  - . escrita sequencial de arquivo

- Primitivas de Manipulação de Diretórios

  . criação de diretório
  . busca inicial em diretório
  . busca em diretório
  . remoção de arquivo (ou diretório)
  . alteração de nome
  . alteração de direito de acesso
  . alteração de password
  . alteração de diretório corrente

- Primitivas de Controle de Acesso

  . bloqueio de registro
  . desbloqueio de registro

3.3.2- A Descrição das Primitivas

Cada primitiva corresponde a uma mensagem genérica ou requisição constituída de um código de comando, seguido de parâmetros adequados. Cada requisição gera uma resposta que contém um código de erro (Ei) seguido de dados quando for o caso.

. ABERTURA DE SESSÃO

parâmetros: nome do usuário
password do usuário

código de retorno: E0 - sucesso
E1 - usuário não reconhecido
E2 - password inválida

Esta primitiva permite proteger o SA contra o acesso de usuários não autorizados. Em caso de sucesso fica estabeleci

da uma conexão entre o usuário e o SA. Em caso de uma conexão anterior estar em atividade, o SA considera implícito um pedido de encerramento da referida sessão. O "diretório corrente" é automaticamente inicializado com o diretório-raiz do usuário.

Este tipo de conexão não deve ser confundido com aquele pré-estabelecido, a nível de transporte, entre o componente físico da rede e o SA, denominado circuito virtual.

. ENCERRAMENTO DE SESSÃO

código de retorno: E0 - sucesso

Esta primitiva desfaz a conexão estabelecida pela primitiva anterior. Os arquivos então abertos para esta conexão são automaticamente fechados, sendo assim liberados os objetos que estiverem bloqueados. Caso não haja conexão esta operação não tem efeito.

. CRIAÇÃO DE ARQUIVO

parâmetros: nome do arquivo
tipo do arquivo
direito de acesso público
tamanho do registro lógico

código de retorno: E0 - sucesso
E1 - diretório ancestral inexistente
E2 - direito de acesso inválido
E3 - arquivo tipo diretório

E4 - arquivo aberto

E5 - número máximo de arquivos excedido

E6 - não há mais canais disponíveis no sistema

E7 - número máximo de entradas excedido

Resultado: canal atribuído ao arquivo

CIA do arquivo

O diretório no qual se deseja criar o arquivo deve existir pois o SA não cria diretórios automaticamente. Caso contrário o SA retorna E1. Uma vez existindo o referido diretório o usuário deverá ter direito de escrita sobre o mesmo, caso contrário retornará E2.

Se o arquivo não existe, ele será criado e automaticamente aberto no modo exclusivo. Caso já exista, ele não pode ser do tipo diretório; caso contrário retornará E3. Em existindo, o arquivo será reinicializado e aberto em modo exclusivo; essa inicialização não ocorrerá portanto se o arquivo estiver em uso, retornando E4.

Caso o arquivo não exista, quando o SA tenta criá-lo existe a possibilidade, embora remota, de não haver mais espaço para a criação de um arquivo, retornando E5.

Quando o SA abre um arquivo, cria para o mesmo um descritor em memória principal. Este descritor contém informações dinâmicas (tais como ponteiros para operações de leitura e/ou escrita) sobre o arquivo e é referenciado por um número denominado canal. Assim sendo, quando dois usuários estiverem compartilhando o mesmo arquivo, serão gerados dois canais diferentes, um para cada usuário. A desativação de um canal pode ser solicitada

através da primitiva de fechamento de arquivos.

Os direitos de acesso próprio são especificados como leitura e escrita podendo ser alterados mais tarde pela primitiva de alteração de direitos de acesso.

. ABERTURA DE ARQUIVO

parâmetros: nome do arquivo ou CIA
modo

código de retorno: E0 - sucesso
E1 - arquivo inexistente
E2 - arquivo tipo diretório
E3 - direito de acesso inválido
E4 - modo conflitante
E5 - não há mais canais disponíveis no sistema.

Resultado: Canal atribuído ao arquivo

Esta primitiva prepara um arquivo já existente para futuras operações de leitura e/ou escrita.

Em caso de sucesso retorna o número do canal lógico que o sistema associou ao arquivo. O usuário deverá referenciar este canal nas operações de entrada e saída subsequentes. O ponteiro de entrada e saída associado àquele canal aponta para o primeiro registro lógico do arquivo.

O SA não permite a abertura de um arquivo tipo diretório, retornando neste caso E2.

Em caso do direito de acesso do usuário sobre o arquivo ser nenhum, o sistema retorna E3.

O parâmetro modo pode solicitar uma abertura em modo exclusivo, em modo leitura ou em modo compartilhado. O SA verifica se o

mesmo arquivo já foi aberto em modo conflitante com o requisitado. Em caso afirmativo retorna E4.

O código de erro E5 já foi examinado por ocasião da descrição da primitiva de abertura de arquivo.

. FECHAMENTO DE ARQUIVO

parâmetro: canal

código de retorno: E0 - sucesso
E1 - canal não aberto

Esta primitiva libera todos os buffers associados com o referido canal e desassocia este canal do arquivo com o qual estava em correspondência. Implicitamente o SA completa as "transactions" envolvidas.

. LEITURA RANDÔMICA DE ARQUIVO

parâmetros: canal
registro inicial
número de registros a serem lidos

código de retorno: E0 - sucesso
E1 - canal não aberto
E2 - final de arquivo
E3 - existe pelo menos um registro bloqueado

Resultado: número de registros lidos
dados lidos

Esta primitiva realiza a leitura de um determinado número de registros lógicos, a partir de um determinado registro lógico especificado.

O SA retorna E1 se o canal referenciado não foi aberto para aquela sessão.

O código E2 será enviado se o registro inicial foi especificado além do final do arquivo não havendo portanto dados a serem lidos.

O número de dados lidos poderá ser menor que a quantidade esperada caso a leitura tenha ultrapassado o final do arquivo,durante a execução da primitiva.

. LEITURA SEQUENCIAL DE ARQUIVO

parâmetros: canal
número de registros a serem lidos

código de retorno: E0 - sucesso
E1 - canal não aberto
E2 - final de arquivo
E3 - existe pelo menos um registro bloqueado

Resultado: número de registros lidos
dados lidos

Esta primitiva é análoga à anterior com a diferença que a leitura se realiza a partir do ponteiro de entrada e saída associado ao canal.

Se o referido ponteiro já se encontra além do final do arquivo, não há dados a serem lidos e o SA retorna E2.

. ESCRITA RANDÔMICA DE ARQUIVOS

parâmetros: canal
registro inicial
número de registros a serem escritos
dados a serem escritos

código de retorno: E0 - sucesso
E1 - canal não aberto

E2 - direito de acesso inválido

E3 - acesso conflitante

E4 - existe pelo menos um registro bloqueado

Se o usuário não tem direito de acesso para escrita sobre o referido arquivo, o SA retorna E2.

Caso o usuário tenha aberto o arquivo em modo leitura, não poderá requisitar operações de escrita recebendo assim o código E3. Pode acontecer também de um (ou mais) registro envolvido na escrita estar bloqueado por outro usuário, retornando-se neste caso o código E4.

. ESCRITA SEQUENCIAL DE ARQUIVO

parâmetros: canal

número de registros a serem escritos

dados a serem escritos

código de retorno: E0 - sucesso

E1 - canal não aberto

E2 - direito de acesso inválido

E3 - acesso conflitante

E4 - existe pelo menos um registro bloqueado

Analogamente ao caso da leitura, a escrita sequencial dispensa a especificação do registro inicial a ser escrito.

. CRIAÇÃO DE DIRETÓRIO

parâmetros: nome do diretório

direito de acesso público

código de retorno: E0 - sucesso

E1 - diretório ancestral inexistente

E2 - arquivo já existente
E3 - direito de acesso inválido
E4 - número máximo de arquivos excedido
E5 - número máximo de entradas excedido

Resultado: CIA do diretório

Analogamente à primitiva de criação de arquivo, o diretório no qual se deseja criar o diretório deve existir; caso contrário retorna E1.

Se o objeto a ser criado já existe o SA retorna E2. O usuário deverá ter direito de escrita sobre o diretório ancestral; caso contrário receberá E3 como código de erro. O direito de acesso próprio é automaticamente considerado para leitura e escrita.

. BUSCA INICIAL EM DIRETÓRIO

parâmetros: nome do diretório de busca ou CIA
nome de referência

Código de retorno: E0 - sucesso
E1 - não existe o diretório especificado para a busca
E2 - direito de acesso inválido sobre o diretório de busca
E3 - não foi encontrado o nome de referência

Resultado: CIA do arquivo
nome de referência encontrado
conteúdo do nodo correspondente da tabe-
ÍNDICE

O SA inicia um processo de busca sobre o referido diretório que passará a ser o "diretório de busca corrente". Se o usuário não tem direito de acesso sobre este diretório, o SA retorna E2.

. BUSCA EM DIRETÓRIO

código de retorno: E0 - sucesso
E1 - não existe o diretório especificado para busca
E2 - direito de acesso inválido sobre o diretório de busca
E3 - não foi encontrado o nome de referência

Resultado: CIA do arquivo
nome de referência
conteúdo do nodo correspondente na tabela ÍNDICE

Esta primitiva é análoga à anterior sendo que a busca continua a partir da última entrada encontrada no diretório de busca corrente.

. REMOÇÃO DE ARQUIVO (OU DIRETÓRIO)

parâmetro: nome do arquivo (ou diretório)

código de retorno: E0 - sucesso
E1 - objeto inexistente
E2 - direito de acesso inválido
E3 - arquivo tipo diretório contém entradas
E4 - arquivo tipo "diretório corrente"
E5 - arquivo já aberto

Esta primitiva envolve a remoção da entrada de diretório que descreve o arquivo (ou diretório) a ser removido. Por isso, o usuário que requisita a operação deve ter direito de acesso para escrita sobre o objeto a remover e sobre o diretório que o descreve; caso contrário o sistema retorna E2.

Os códigos E3 e E4 só têm sentido quando o objeto é do tipo diretório.

Quando o diretório a ser removido referencia ainda algum arquivo, ele não poderá ser removido, retornando E3.

Quando o diretório a ser removido é diretório corrente para algum usuário, ele não será removido, retornando E4.

Se o arquivo a ser removido estiver aberto o pedido será recusado retornando E5.

. ALTERAÇÃO DE NOME

parâmetros: nome do arquivo (ou diretório)
novo nome de referência

código de retorno: E0 - sucesso
E1 - objeto inexistente
E2 - direito de acesso inválido
E3 - arquivo tipo "diretório corrente"
E4 - arquivo já aberto
E5 - o novo nome já existe

O usuário que requisita a operação deve ter direito de acesso para escrita sobre o objeto, caso contrário, o sistema retorna E2. Se o objeto for do tipo diretório, não pode ser diretório corrente para nenhuma conexão; se isso acontecer, o SA retorna E3.

Se o objeto for um arquivo não diretório o mesmo não poderá estar aberto; caso contrário, o SA retorna E4.

. ALTERAÇÃO DE DIREITO DE ACESSO

parâmetros: nome do arquivo (ou diretório) ou CIA
novo direito de acesso próprio
novo direito de acesso público

código de retorno: E0 - sucesso
E1 - objeto inexistente
E2 - direito de acesso inválido
E3 - arquivo já aberto
E4 - arquivo tipo "diretório corrente"

Apenas o usuário que criou este objeto tem o direito de fazer este tipo de operação; os demais receberão E2 como código de retorno.

Um usuário não poderá realizar este tipo de operação se o arquivo em questão estiver aberto de modo a não gerar inconsistências para outros usuários; o código de retorno correspondente é E3.

Se o arquivo for tipo diretório, o SA retornará E4 caso ele seja diretório corrente para algum canal.

. ALTERAÇÃO DE PASSWORD

parâmetro: Nova password

código de retorno: E0 - sucesso

Esta primitiva permite a troca de uma password de usuário por outra, sendo portanto um apoio à segurança do sistema.

. ALTERAÇÃO DE DIRETÓRIO CORRENTE

parâmetro: nome do diretório ou CIA

. código de retorno: E0 - sucesso
E1 - diretório inexistente
E2 - direito de acesso inválido

Esta primitiva especifica o diretório indicado para diretório corrente da conexão. Quando o usuário não tem direito de acesso sobre tal diretório, o código E2 retorna como resposta.

. BLOQUEIO DE REGISTRO

parâmetros: canal
número do registro inicial a ser blo - queado
número de registros a serem bloqueados

código de retorno: E0 - sucesso
E1 - canal não aberto
E2 - canal não aberto em modo com - partilhado
E3 - existe pelo menos um registro já bloqueado
E4 - existe pelo menos um registro fora do arquivo

Esta primitiva bloqueia um conjunto de registros consecutivos de modo que nenhum outro canal poderá bloqueá-los ou acessá-los até que sejam desbloqueados. É necessário que o arquivo tenha sido aberto em modo compartilhado; caso contrário, o sistema retornará E2.

Se o usuário tentar bloquear uma região inexistente do arquivo, o SA responde com E4.

. DESBLOQUEIO DE REGISTRO

parâmetro: canal

código de retorno: E0 - sucesso

E1 - canal não aberto

E2 - canal não aberto em modo compartilhado

Esta primitiva desbloqueia o conjunto de registros bloqueados pela primitiva de bloqueio. Também só pode ser empregada para arquivos que foram abertos em modo compartilhado; caso contrário retorna E2.

3.3.3- Um Exemplo de Sessão

A figura III.8 ilustra um exemplo de utilização dos serviços disponíveis. Após o estabelecimento de um circuito virtual com o SA, o usuário requisita uma conexão lógica (sessão) ao SGA, realiza operações sobre um objeto e solicita o encerramento da conexão.

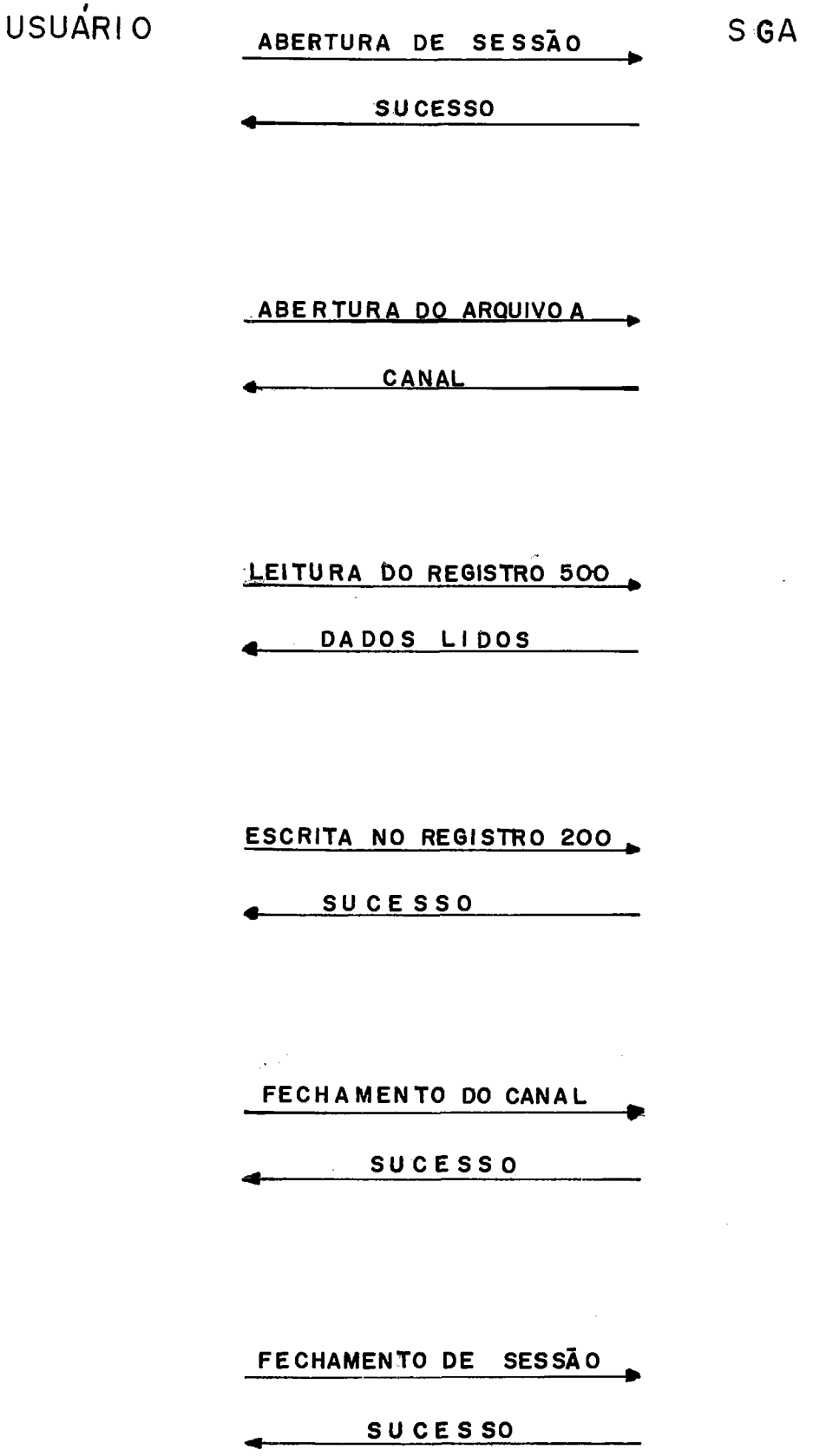

USUÁRIO
SGA
ABERTURA DE SESSÃO
SUCESSO
ABERTURA DO ARQUIVO A
CANAL
LEITURA DO REGISTRO 500
DADOS LIDOS
ESCRITA NO REGISTRO 200
SUCESSO
FECHAMENTO DO CANAL
SUCESSO
FECHAMENTO DE SESSÃO
SUCESSO

FIGURA III-8

CAPÍTULO IV

IMPLEMENTAÇÃO

Este capítulo define a estrutura de dados e os algoritmos utilizados na implementação dos serviços oferecidos pelo SGA, usando o PASCAL como linguagem de referência.

4.1- Descrição Geral

O SGA mantém um processo para cada circuito virtual estabelecido entre o SA e um determinado usuário. Os processos resultantes concorrem entre si no acesso aos recursos comuns: "subsistema de comunicação" e "dispositivo de memória de massa". A manutenção dos diversos circuitos virtuais em paralelo otimiza desta forma a utilização dos recursos por parte dos usuários.

Todos os processos compartilham entre si um código comum reentrante que descreveremos ao longo deste capítulo. A sequência de execução do referido programa é mostrada na figura IV.1.

O programa principal decodifica a requisição enviada pelo circuito virtual em questão e ativa a função de serviço correspondente à primitiva solicitada. Toda vez que a função de serviço necessita realizar uma operação sobre uma estrutura residente em memória de massa ativa por sua vez uma rotina de manipulação de objetos.

Procederemos aqui a uma descrição da manipulação das pricipais estruturas utilizadas pelo SGA: o nodo e as tabelas Ativa e Canal. Esta descrição, embora formalizada mais tarde, facilitará a compreensão dos próximos itens.

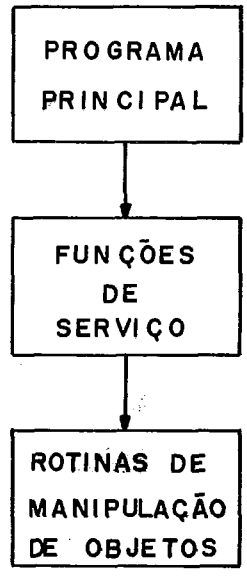


FIGURA IV·1

A tabela ÍNDICE é um vetor de descritores de arquivos ou nodos. Cada descritor representa o estado de um arquivo, e é descrito por um record chamado TipoNodo. Outras estruturas de dados, residentes na memória, são compostas de elementos de TipoNodo.

Cada vez que um arquivo é aberto, o conteúdo do seu nodo é copiado para uma tabela de nodos ativos na memória principal, denominada TabelaAtiva. A tabela de nodos ativos contém ainda informação sobre o modo para o qual o arquivo foi aberto, controlando assim o acesso ao mesmo. Diversos usuários podem acessar paralelamente o mesmo arquivo (modo não-exclusivo) através de diferentes canais. O vetor TabelaCanal guarda informações sobre cada canal, tais como, o registro corrente de acesso ao arquivo e a referência para a TabelaAtiva. Uma entrada na TabelaAtiva só é liberada quando não houver nenhuma referência na TabelaCanal para esta entrada. Enquanto o arquivo se encontra aberto, o sistema proibe alteração do seu nome ou seus di -

reitos de acesso bem como impede que ele seja removido.

A primitiva de criação de arquivo cria um novo nodo na tabela ÍNDICE, grava o CIA e o nome no diretório correspon - dente e então procede a uma operação de abertura. As operações de leitura e escrita apenas consultam e atualizam as tabelas na memória principal sem criar novas entradas. A figura IV.2 ilustra a correspondência entre essas tabelas.

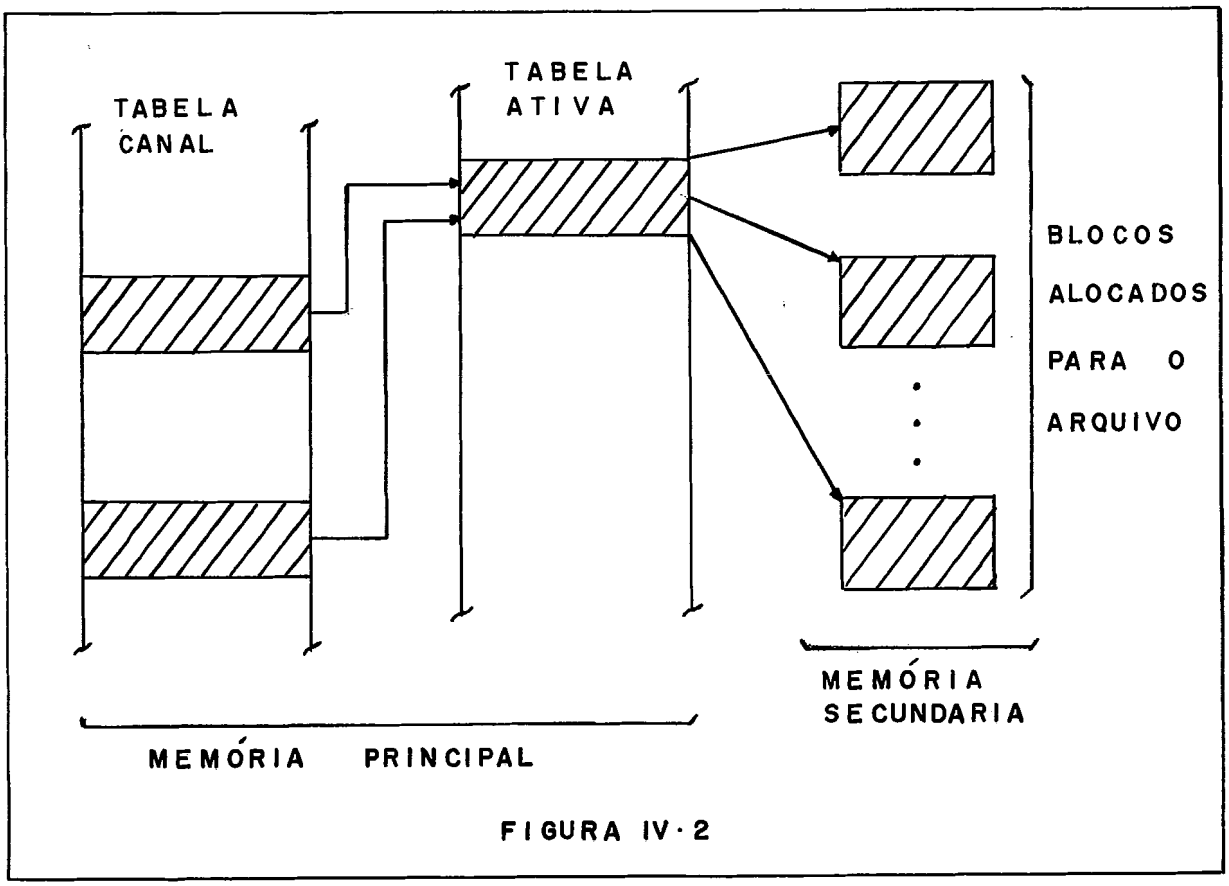


FIGURA IV·2

Este capítulo foi organizado de uma forma semelhante a um texto PASCAL e contém: apresentação da estrutura de dados (constantes, tipos, estruturas e variáveis globais); definição das funções de manipulação de objetos; descrição dos algoritmos das funções de serviço e do programa principal.

4.2- A Definição das Principais Constantes

```
const
    CvMax = (* número máximo de circuitos virtuais *);
    AtivaMax = (* número de entradas da TabelaAtiva *);
    CanalMax = (* número de entradas da TabelaCanal *);
    CiaMax = (* número de entradas da TabelaÍndice *);
    CiaRaiz = (* CIA da RAIZ *);
    Eof - (* indicativo de final de arquivo *);
    EntradaMax = (* número máximo de entradas em um diretó
                 rio *);
    RegMax = (* número máximo de registros lógicos num ar
              quivo *);
    NumRegMax = (* número máximo de registros especificado
                 nas primitivas de leitura e/ou escrita *);
    NumRegBloqMax = (* número máximo de registros bloqueados
                    por uma primitiva de bloqueio *);
    TamBloco = (* tamanho do bloco em bytes *);
    N = (* número máximo de blocos alocados para um arquivo
        e referenciados dentro do nodo *);
    M = (* número máximo de blocos que podem ser referencia-
        dos por um bloco de acesso indireto *);
    NumBlocoMax - (* número de blocos existentes na memória
                  secundária *);
    CacheMax = (* número de blocos existentes na região de
                memória principal reservada para cache *);
    CadeiaMax = (* número máximo de caracteres por cadeia
                 alfanumérica *);
    PrimitivaMax = (* número de primitivas disponíveis no
                   SGA *);
```

4.3- A Definição dos Principais Tipos Simples

```
type
    Modos = (ExclusivoNormal, ExclusivoEspecial,
             ApenasLeitura, Compartilhado);
    Objetos = (Arquivo, Diretório , DiretórioRaiz);
    Direitos = (Nenhum, Leitura, Escrita);
    TipoCia = 0..CiaMax;
    TipoEntrada = 0..EntradaMax;
    TipoCanal = 0..CanalMax;
    TipoAtiva = 0..AtivaMax;
    TipoRegistro = 0..RegMax;
    TipoBloco = 0..NumBlocoMax;
    Byte = 0..255;
```

4.4- A Definição dos Principais Tipos Estruturados

```
Type
    Cadeia = array[0..CadeiaMax] of char;
    TipoBuffer = array[0..TamBloco] of Byte;
    TipoNodo = record
        Dono : TipoEntrada;
        Tipo : Objetos;
        DireitoPróprio : Direitos;
        DireitoPúblico : Direitos;
        Tamanho : 0..RegMax;
        RegistroLógico : 0..TamBloco
        Mapa : array[1..N] of TipoBloco;
        Próximo1 : TipoBloco;
        Próximo2 : TipoBloco
    end;
```

Descrição:

Esta estrutura é a imagem de um nodo.

O campo Dono contém a entrada da RAIZ correspondente ao dono do arquivo. É este campo que indica quando o nodo está livre.

O campo Tipo indica o tipo do arquivo: diretório, diretório-raiz ou não-diretório.

Os campos DireitoPróprio e DireitoPúblico correspondem respectivamente aos direitos de acesso próprio e público.

O campo Tamanho indica o tamanho do arquivo, medido em número de registros lógicos.

O campo RegistroLógico recebe o tamanho do registro lógico, especificado pelo usuário no momento da criação do arquivo, não podendo ultrapassar o tamanho de um bloco.

O campo Mapa contém N entradas que apontam para os primeiros N blocos alocados para o arquivo. Se o arquivo for maior que N blocos, então o campo Próximo1 aponta para um bloco que referencia os próximos M blocos do arquivo (figuraIII.4). Caso N + M blocos não sejam suficientes, o campo Próximo2 aponta para um outro bloco de acesso indireto.

```
TipoAtiva = record
    Cia : TipoCia;
    Modo : Modos;
    Cache : TipoBloco;
    ApontadorCache : 0..CacheMax;
    Contador :0..CanalMax;
    Nodo : TipoNodo
end;
```

Descrição:

Esta estrutura é a imagem de uma entrada na TabelaAtiva. Cada vez que um novo arquivo é aberto, informações sobre o mesmo são mantidas em uma das entradas da referida tabela.

O campo Cia recebe o CIA do arquivo. Modo indica o modo segundo o qual o arquivo foi aberto.

O campo Cache indica se o arquivo em questão contém um bloco em cache; em caso positivo, o próprio campo dá o endereço do bloco e o seguinte(ApontadorCache) determina a localização do referido bloco na memória principal.

O campo Contador contabiliza o número de referências que a Tabela Canal faz àquela entrada da TabelaAtiva. Cada vez que um novo canal é criado para aquele arquivo, este contador é incrementado; analogamente quando o canal é desativado, o contador é decrementado. Quando o contador atinge o valor nulo,a entrada correspondente da TabelaAtiva é liberada e o campo Cia passa a indicar este fato.

O campo Nodo contém todas as informações existentes no nodo, conforme a descrição da estrutura TipoNodo.

```
TipoCanal = record
    IndAtiva : TipoAtiva;
    RegCorrente : TipoRegistro;
    RegBloq : TipoRegistro;
    NumRegBloq : 0..NumRegBloqMax;
    Cv : 0..CvMax
end;
```

Descrição:

Esta estrutura descreve uma entrada da TabelaCanal.

O primeiro campo aponta para a entrada da TabelaAtiva que descreve o arquivo em questão.

O campo RegCorrente é inicializado com zero quando o arquivo é aberto e vai sendo atualizado de acordo com as operações de leitura e escrita realizadas sobre aquela imagem do arquivo.

Os campos RegBloq e NumRegBloq só têm sentido quando o arquivo for aberto em modo compartilhado. Neste caso, o campo NumRegBloq é inicializado com zero até que o usuário requisita uma operação de bloqueio sobre o referido canal. O campo RegBloq é então preenchido com o registro inicial bloqueado e o campo NumRegBloq recebe o número de registros bloqueados. Um pedido de desbloqueio reinicializa o campo $NumReg_{Bloq}$ com zero.

O campo Cv indica o circuito virtual para o qual o canal foi aberto. Em caso de encerramento de sessão, por exemplo, o SGA deve liberar todas as entradas da TabelaCanal que estiverem alocadas para o circuito virtual que solicitou a operação; para tal, realiza uma busca sobre a TabelaCanal consultando o campo Cv.

4.5- A Definição das Principais Variáveis Globais

```
var
    Cv : 0..CvMax;
```

Descrição: Circuito virtual que solicitou a operação a ser decodificada pelo programa principal.

```
Comando: 0..PrimitivaMax;
```

Descrição: Esta variável define a primitiva solicitada pelo circuito Cv.

```
Conexão: array[0..CvMax] of TipoEntrada;
```

Descrição: O vetor Conexão descreve a situação de cada um dos circuitos virtuais suportados pelo SA. Quando um usuário estabelece uma conexão lógica com o SGA, através da primitiva de abertura de sessão, a entrada que ele ocupa na RAIZ é anotada neste vetor, na posição correspondente ao circuito virtual que ele estiver utilizando (Conexão [Cv]).

```
TabelaCanal: array[0..CanalMax] of TipoCanal;

TabelaAtiva: array[0..AtivaMax] of TipoAtiva;

CiaDirCorrente : array[0..CvMax] of TipoCia;
```

Descrição: A cada conexão lógica fica associado um "diretório corrente" cujo CIA é anotado neste vetor. No momento da abertura de sessão a entrada correspondente é inicializada com o CIA do diretório-raiz do usuário. O usuário pode ir alterando este valor através da primitiva de alteração de diretório corrente.

CiaDirBuscaCorrente : array[0..CvMax] of TipoCia;

Descrição: A cada conexão lógica fica também associado um "diretório de busca corrente" sobre o qual o usuário poderá utilizar a primitiva de busca em diretório. O CIA do referido diretório é colocado na entrada correspondente à conexão. No momento da abertura da conexão, este valor não é inicializado. O usuário deverá portanto inicializá-lo com a primitiva de busca inicial.

EntradaBuscaCorrente : array[0..CvMax] of TipoEntrada;

Descrição: Para cada conexão, este valor indica a entrada do "diretório de busca corrente" a partir da qual será realizada a próxima busca em diretório. A "busca inicial" sempre é realizada a partir da primeira entrada. Uma vez bem sucedida uma busca, esta variável fica apontando para a entrada seguinte. Em caso de insucesso, o usuário deverá reinicializá-la através da primitiva de busca inicial.

ReferênciaBuscaCorrente : array[0..CvMax] of Cadeia;

Descrição: Durante uma busca o SGA procura pelo nome de referência especificado na entrada correspondente à conexão, inicializada pela primitiva de busca inicial.

4.6- As Rotinas de Manipulação de Objetos

Estas rotinas são responsáveis por tarefas específi-

cas de manipulação do conteúdo da memória de massa, tais como: buscas em diretórios, gravação de atributos, atualização de arquivos, leitura e escrita em registros, etc.

Quando a operação solicitada envolve alteração em alguma região estrutural da memória (nodos, tabelas de alocação e entradas de diretórios) a rotina aciona uma "transaction" utilizando os conceitos de atomicidade e listas de intenções descritos no capítulo anterior. As operações de escrita sobre essas estruturas e sobre as listas de intenções são realizadas em memória estável.

Segue-se a definição das referidas rotinas. As primeiras se limitam a realizar apenas operações de leitura na memória secundária; as demais envolvem alteração propriamente dita.

```
- procedure BuscaV (Arquivo : Cadeia ; var CiaArq : TipoCia;
  var CiaAncestral : TipoCia; var EntradaAncestral :
  TipoEntrada);

  begin
       (* . recebe o nome de um arquivo (Arquivo)
          . executa a busca do referido arquivo,     denominada
            "busca vertical"
          . em caso de sucesso retorna:
            -o CIA do arquivo (CiaArq)
            -o CIA do diretório ancestral (CiaAncestral)
            -a entrada do diretório ancestral que   corresponde
             ao arquivo (EntradaAncestral)
          . em caso de insucesso retorna CiaArq = 0 *)

  end ;
```

```
- procedure BuscaH (Referência : Cadeia; CiaDir : TipoCia;
  var Entrada : TipoEntrada; var CiaArq : TipoCia);

  begin
       (* . recebe o nome de referência de um arquivo (Referên-
            cia) e o CIA de um diretório (CiaDir)
          . realiza a busca do nome de referência sobre o dire-
            tório, denominada "busca horizontal"
          . em caso de sucesso retorna:
            - a entrada do diretório que corresponde ao arqui-
              vo (Entrada)
            - o CIA do arquivo (CiaArq)
          . em caso de insucesso retorna CiaArq = 0 *)

  end;

- procedure Leitura (Canal : TipoCanal; Registro : TipoRegistro;
  NumReg : 0..NumRegMax; var NumRegLidos : 0..NumRegMax;
  Buffer : Tipo Buffer);

  begin
       (* . recebe um arquivo já aberto em Canal
          . realiza a leitura de NumReg registroslógicos a par-
            tir do registro Registro
          . o número real de registros lidos é colocado em
            NumRegLidos
          . os dados lidos são colocados em Buffer
          . o valor do registro corrente na TabelaCanal é auto-
            maticamente atualizado *)

  end;
```

- procedure GravaPassword (Password : Cadeia);

  begin

  (* . recebe uma password em Password

  . grava a password na entrada da RAIZ, correspondente ao usuário (dada por Conexão [Cv]) *)

  end;

- procedure GravaDireitos (Cia : TipoCia; DireitoPróprio, DireitoPúblico : Direitos);

  begin

  (* . recebe o CIA de um arquivo (Cia)

  . recebe os novos direitos de acesso (DireitoPróprio e DireitoPúblico)

  . grava os novos direitos no nodo correspondente ao Cia *)

  end;

- procedure GravaReferencia (CiaDir : TipoCia; Entrada : TipoEntrada; Referência : Cadeia);

  begin

  (* . recebe o CIA de um diretório (CiaDir) e uma entrada (Entrada) deste mesmo diretório

  . recebe um nome de referência (Referência)

  . grava este nome de referência na referida entrada *)

  end;

- procedure GravaNovaEntrada (CiaDir, Cia : TipoCia; Referência : Cadeia; Nodo : TipoNodo; var Entrada : TipoEntrada);

```
begin;
      (* . recebe o CIA de um diretório (CiaDir)
         . recebe o CIA correspondente a um nodo livre (Cia)
         . recebe todas as informações a serem gravados no
           nodo na variável Nodo
         . busca uma entrada livre no diretório definido por
           CiaDir
         . em caso de não haver entrada livre, retorna Entra
           da com um valor inválido
         . em caso de haver entrada livre, grava na mesma um
           nome de referência (Referência) e o CIA do nodo
           livre (Cia); grava no nodo as informações recebidas
           em Nodo; retorna a entrada encontrada em Entra
           da *)

end;

- procedure ReinicializaEntrada (CiaDir : TipoCia;
  Entrada : TipoEntrada; Referência : Cadeia; Nodo : TipoNodo);

  begin
       (* . recebe o CIA de um diretório (CiaDir)
          . recebe uma entrada a ser reinicializada neste dire-
            tório (Entrada)
          . recebe todas as informações, a serem gravadas no no
            do pertencente a esta entrada, na variável Nodo
          . recebe o novo nome de referência a ser gravado nes-
            ta entrada (Referência)
          . grava o nome de referência na entrada
          . grava no nodo, as informações recebidas em Nodo *)

  end;
```

```
- procedure RemoveEntrada (CiaDir : TipoCia;
  Entrada : TipoEntrada);

  begin
       (* . recebe o CIA de um diretório (CiaDir)
          . recebe uma entrada a ser removida neste diretório
          . remove a referida entrada e libera o nodo correspon-
            dente, assim como todos os blocos alocados para es
            te arquivo *)

  end;

- procedure Atualiza (Canal : TipoCanal);

  begin
       (* . recebe um arquivo aberto em Canal
          . atualiza o nodo correspondente ao arquivo a partir
            das informações contidas na TabelaAtiva
          . atualiza o mapade alocação de acordo com as modifi-
            cações geradas pelas operações de escrita sobre es-
            te arquivo *)

  end;

- procedure Escrita (Modo : Modos; Canal : TipoCanal;
  Registro : TipoRegistro ; NumReg : 0..NumRegMax;
  var Buffer : TipoBuffer);

  begin
       (* . recebe um arquivo aberto em Canal, no modo Modo
          . realiza a escrita dos dados recebidos em Buffer,nos
            NumReg registros lógicos, localizados a partir de
            Registro.
          . o valor do registro corrente na TabelaCanal assim
            como o tamanho do arquivo e o mapa do arquivo na
            TabelaAtiva são automaticamente atualizados
          . em caso de modo exclusivo especial, a escrita sobre
```

registros já existentes é realizada sobre blocos livres que, apenas por ocasião do fechamento do arquivo, serão definitivamente incorporados para o mesmo *)

end;

4.7 As Funções de Serviço

Descreveremos aqui o procedimento lógico desencadeado pela ativação das principais primitivas. Toda vez que tal procedimento envolver uma operação sobre um objeto residente em memória de massa, será então ativada uma das rotinas de manipulação de objetos, definidas no item 4.6.

Para tornar a leitura mais simples, vamos descrever somente as primitivas de abertura de arquivo, leitura randômica, escrita randômica e fechamento de arquivo.

O procedimento relativo às demais primitivas se encontra no Anexo III.

- ABERTURA DE ARQUIVO

```
function  AbreArq (Cia : TipoCia; Modo : Modos;
var Canal : TipoCanal) :E0..E5;

begin
     if (* o nodo correspondente ao CIA está livre *)
        then return (E1); (* arquivo inexistente *)
     if (* arquivo é tipo diretório ou diretório-raiz *)
        then return (E2); (* arquivo tipo diretório *)
     if (* o usuário não tem nenhum direito de acesso sobre o
           arquivo *)
        then return (E3); (* direito de acesso inválido *)
     (* busca entrada livre na TabelaCanal *)
```

```
      if (* busca sem sucesso *)
         then return (E5); (* não há canais livres *)
      (* verifica se o arquivo já estava aberto,  consultando
         a TabelaAtiva *)
      if (* o arquivo já estava aberto *)
         then begin
              if (* o usuário deseja abrir em modo exclusivo *)
                 then return (E4); (* modo conflitante *)
              if (* o usuário deseja abrir em modo  diferente
                 do já escolhido *)
                 then return (E4); (* modo conflitante *)
              (* incrementea o campo Contador da TabelaAtiva *)
              end
         else begin
              (* preenche os campos de uma entrada livre  da
                 TabelaAtiva com informações sobre o  arquivo
                 e Contador = 1 *)
              end ;
      (* preenche os campos da TabelaCanal com IndAtiva apon -
         tando para a TabelaAtiva, RegCorrente = 0 e
         NumRegBlog = 0 *)
      (* Canal : = entrada da TabelaCanal *)
      return (E0)

   end;
```

- LEITURA RANDÔMICA DE ARQUIVO

```
   function  LeituraRandom (Canal : TipoCanal;
   Registro : TipoRegistro; NumReg : 0..NumRegMax;
   var NumRegLidos : 0..NumRegMax; var Buffer :TipoBuffer) :
   E0..E3;
```

```
begin
     if (* canal não está aberto *)
        then return (E1); (* canal não aberto *)
     if (* registro inicial de leitura (Registro) está  fora
        do arquivo *)
        then begin
             TabelaCanal [Canal]. RegCorrente : = Eof;
             return (E2); (* final de arquivo *)
             end;
     if (* modo de abertura é Compartilhado *)
        then begin
             if (* pelo menos um dos registros a serem  lidos
                   está bloqueado por outro canal *)
                then return (E3) (* registro bloqueado *)
             end;
     (* realiza a leitura propriamente dita (Leitura) *)
     return (E0) (* sucesso *)
end;
```

- ESCRITA RANDÔMICA DE ARQUIVO

```
function EscritaRandom (Canal : TipoCanal;
Registro : TipoRegistro; NumReg : NumRegMax;
Buffer : TipoBuffer) : E0..E4;

begin
     if (* canal não está aberto *)
        then return (E1); (* canal não aberto *)
     if (* o usuário não tem direito de escrita sobre o arqui
           vo *)
        then return  (E2); (* direito de acesso inválido *)
     if (* modo de abertura é ApenasLeitura *)
        then return (E3); (* modo de acesso conflitante  *)
```

```
        if (* modo de abertura é Compartilhado *)
           then begin
                if (* pelo menos um dos registros a serem escri-
                      tos está bloqueado por outro canal *)
                then return (E4) (* registro bloqueado *)
                end;
        (* realiza a escrita propriamente dita (Escrita) *)
        return (E0)
    end;
```

- FECHAMENTO DE ARQUIVO

```
    function FechaArq (Canal : TipoCanal) : E0..E1;

    begin
        if (* canal não está aberto (entrada livre na TabelaCanal) *)
           then return (E1); (* canal não aberto *)
        if (* modo de abertura é diferente de ApenasLeitura *)
           then (* atualiza nodo e mapa de alocação em memória
                   secundária (Atualiza) *)
        (* remove canal da TabelaCanal *)
        (* decrementa Contador na TabelaAtiva *)
        if (* Contador = 0 *)
           then (* Remove entrada da TabelaAtiva *)
        return (E0) (* sucesso *)
    end;
```

4.8- O Programa Principal

É a seguinte a estrutura do programa principal relativo a cada circuito virtual.

```
begin
     white true do
     begin
          (* espera  comando do subsistema de transporte *)
          case Comando of
               0 : ...;
               1 : ...;
                    i: begin
                         (* espera parâmetros do subsistema de
                            transporte *)
                         (* realiza a análise dos  parâmetros
                            recebidos *)
                         (* ativa a função de serviço  ade -
                            quada *)
                         (* formata os resultados *)
                         (* transmite os resultados para o sub
                            sistema de transporte *)
                       end;
               PrimitivaMax: ...
          end
     end
end.
```

Descrição: Cada "label" do comando "case" corresponde à execução da função associada à primitiva definida pela variável Comando.

Após receber a requisição enviada pelo usuário, o SGA realiza a crítica dos parâmetors envolvidos e ativa a função de serviço adequada. A seguir, formata os resultados, transmitindo-os para o subsistema de transporte.

4.9- O Gerenciamento dos Processos

O gerenciamento dos processos e o controle de acesso aos recursos são realizados pelo sistema operacional (figura IV.3).

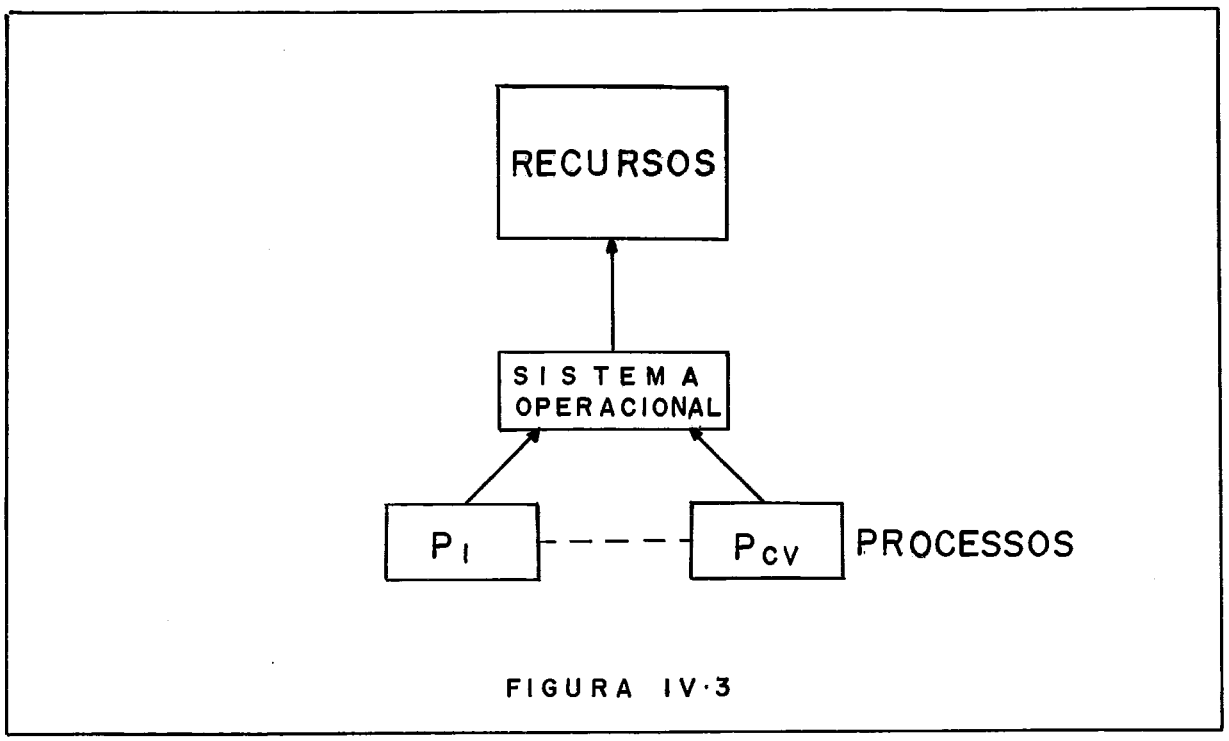


FIGURA IV·3

Um escalador é responsável por multiplexar o processador entre os processos, considerando-os igualmente prioritários.

O controle de acesso aos recursos é realizado por monitores específicos, responsáveis respectivamente por: transmissão de mensagem para o usuário, recepção de mensagem do usuário e manipulação do dispositivo físico de memória.

Quando um processo Pi, correspondente ao circuito virtual i, deseja receber dados do subsistema de transporte, ativa o monitor de recepção passando como parâmetros: o número de dados a receber, o endereço de memória onde deverão ser colocados os dados, e um limite de tempo máximo (podendo ser infini

to) para a chegada da mensagem. O monitor armazena o pedido, colocando o processo requisitante em estado de espera do recurso. Uma vez recebida a mensagem ou detetado algum erro de comunicação, o processo é marcado como pronto para ser eventualmente recolocado em execução pelo escalador. A chamada ao monitor retorna para o processo um resultado informando sobre o sucesso da operação.

Analogamente, o monitor de transmissão transmite para o usuário uma mensagem especificada pelo processo Pi. Para tal, recebe como parâmetros o número de dados a transmitir e o endereço de memória onde se encontram os referidos dados. Uma vez confirmada pelo susbistema de transporte a recepção do último pacote relativo àquela mensagem ou em detetando algum erro, o sistema recoloca o processo Pi como pronto para executar, retornando o resultado da operação.

O monitor dedicado às operações em disco recebe como parâmetros o tipo de operação a ser executada (leitura, escrita, posicionamento de cabeça, formatação, testes, etc.) e os parâmetros correspondentes. Retorna para o processo requisitante o resultado obtido.

Os monitores mantêm listas onde são armazenados os pedidos de recursos realizados pelos diferentes processos.

CAPÍTULO V

AVALIAÇÃO

Neste capítulo descreveremos algumas experiências realizadas sobre o protótipo em laboratório, no sentido de identificar os efeitos de alguns parâmetros de projeto sobre o desempenho do sistema. As medidas de tempo obtidas serão então interpretadas e avaliadas.

As experiências aqui descritas correspondem a um teste de carga, não revelando o comportamento do protótipo em um ambiente computacional real.

Três aspectos principais influenciam as operações de leitura e escrita requisitadas pelo usuário: o tempo de acesso do dispositivo utilizado, o número de acessos necessários à localização do registro lógico através da estrutura do sistema e a existência de caches.

A fim de isolar os referidos efeitos selecionamos uma sequência ordenada de operações a serem requisitadas por cada usuário do SA. Este conjunto de operações foi denominado por nós de programa de teste.

5.1- Descrição do Programa de Teste

Este programa se inicia abrindo um arquivo em modo exclusivo normal. Prossegue então em duas fases: leitura e escrita sobre o referido arquivo. Cada uma dessas fases se subdivide por sua vez em quatro outras, cada uma delas acessando de modo particular uma determinada região do arquivo: acesso sequencial à primeira metade do arquivo; acesso sequencial à segunda parte do arquivo; acesso randômico à primeira parte do

arquivo; e acesso randômico à segunda parte do arquivo.A figura V.1, ilustra a seqüência de execução do programa.

Como podemos observar, o arquivo foi logicamente dividido em duas partes de modo que a primeira fica totalmente contida nos blocos referenciados dentro do próprio nodo, sendo portanto de acesso mais rápido. Já a manipulação de um registro lógico pertencente à segunda parte depende de um acesso indireto adicional para a localização do mesmo. Esta divisão permite avaliar o "overhead" envolvido no referido processo de localização através da estrutura do SA.

Os dois métodos de acesso utilizados (sequencial e randômico) procuram evidenciar o efeito do uso da memória cache. No caso sequencial um único acesso a disco corresponderá à leitura de diversos registros lógicos contidos no bloco lido e mantido em memória. No acesso randômico, o programa de teste obriga que duas leituras consecutivas utilizem blocos diferentes, impedindo propositalmente o aproveitamento do cache.

5.2- Descrição da Experiência

O programa foi executado simultaneamente por diferentes usuários ligados à rede. O teste foi repetido variando-se parâmetros tais como o número de usuários e a posição relativa dos respectivos arquivos. Foram medidos os tempos envolvidos na execução das diferentes fases do programa para um determinado usuário.

No SA foi utilizado um disco de 5M bytes, arbitrando-se blocos de 512 bytes e um máximo de 1024 nodos na tabela ÍNDICE. Os arquivos dos usuários foram criados com registros de 128 bytes. Um bloco suporta neste caso 4 registros lógicos. Como um nodo referencia diretamente até 15 blocos, cada metade

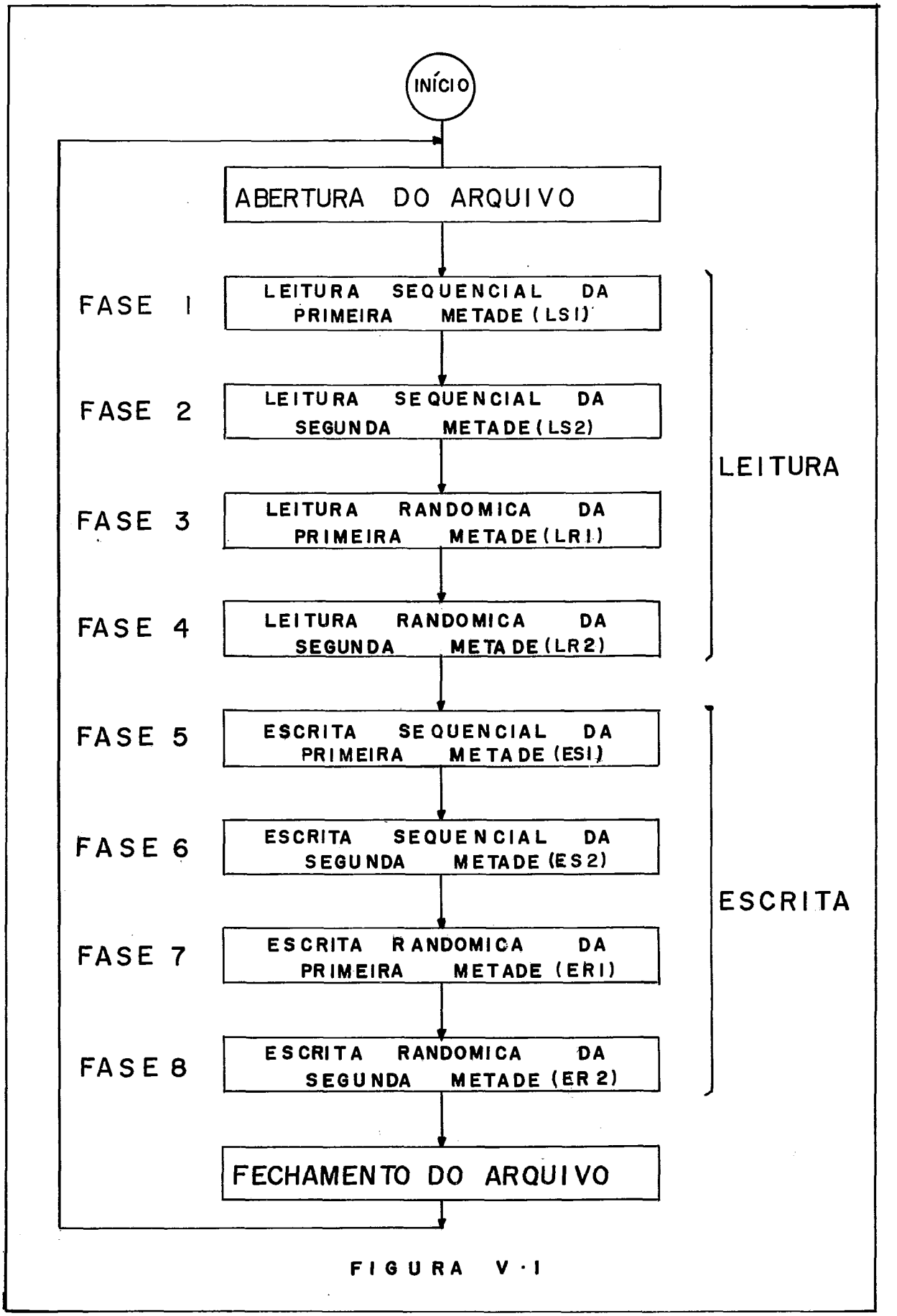

INÍCIO
ABERTURA DO ARQUIVO
FASE 1
LEITURA SEQUENCIAL DA PRIMEIRA METADE (LS1)
FASE 2
LEITURA SEQUENCIAL DA SEGUNDA METADE (LS2)
FASE 3
LEITURA RANDOMICA DA PRIMEIRA METADE (LR1)
FASE 4
LEITURA RANDOMICA DA SEGUNDA METADE (LR2)
LEITURA
FASE 5
ESCRITA SEQUENCIAL DA PRIMEIRA METADE (ES1)
FASE 6
ESCRITA SEQUENCIAL DA SEGUNDA METADE (ES2)
FASE 7
ESCRITA RANDOMICA DA PRIMEIRA METADE (ER1)
FASE 8
ESCRITA RANDOMICA DA SEGUNDA METADE (ER2)
ESCRITA
FECHAMENTO DO ARQUIVO

FIGURA V·1

do arquivo corresponde a 60 registros lógicos.

O "overhead" de comunicação através da rede foi desprezado em relação aos valores obtidos nas medidas.

5.3- Interpretação dos Resultados

Para facilitar a análise dos resultados, os testes foram divididos em etapas:

5.3.1- Primeira Etapa

O programa foi executado em respectivamente um, dois e três usuários interligados através da rede. Cada um dos usuários trabalha sobre um arquivo diferente, aberto em modo exclusivo-normal. Em cada caso, os arquivos envolvidos(ARQ1, ARQ2 , ARQ3) foram colocados em posições diferentes, distantes respectivamente de 130K bytes, 1M bytes e 2M bytes. A figura V.2 ajuda a visualizar as condições de teste.

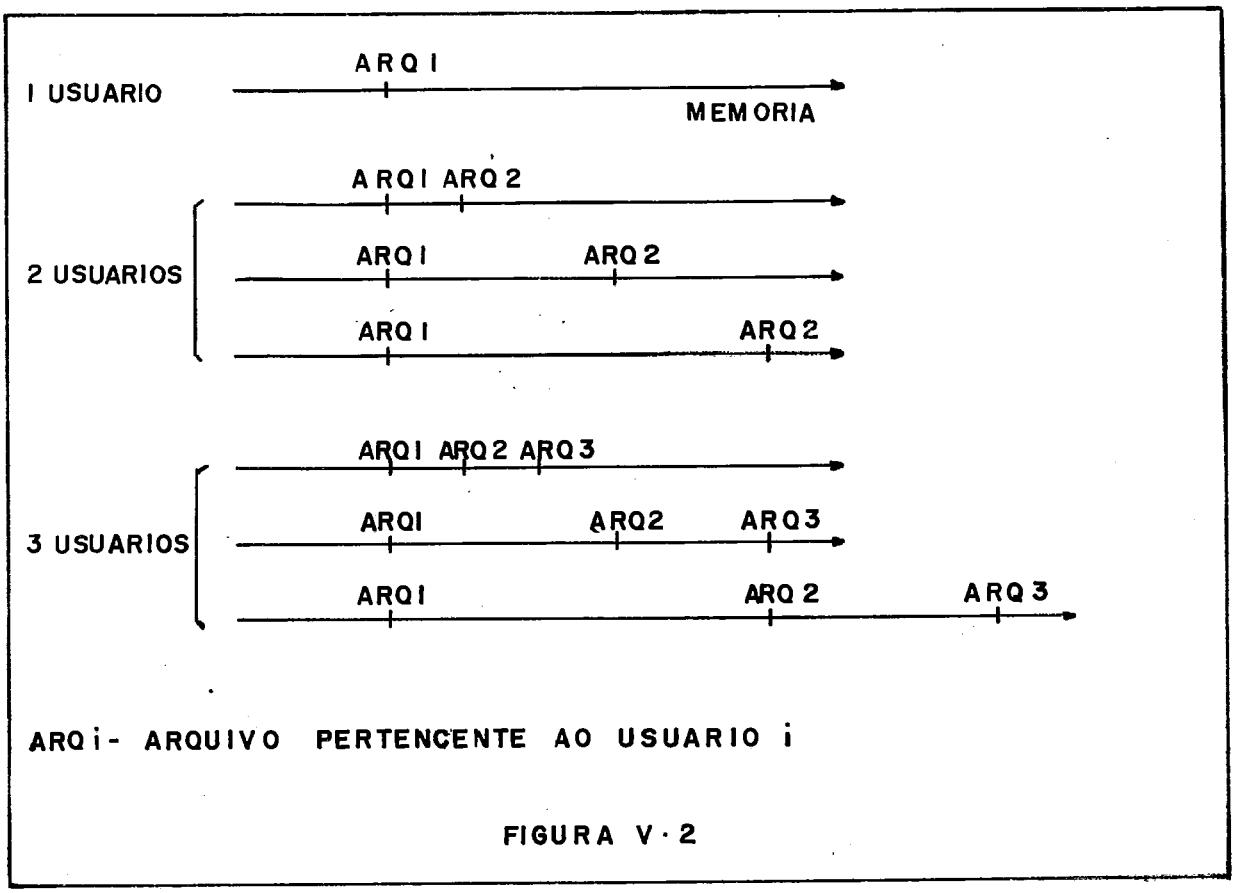


FIGURA V·2

Nesta primeira etapa, visando facilitar o isolamento dos efeitos a serem avaliados o SA atende os usuários um a um em sequência ("round robin"). Esta é a situação mais desfavorável em termos de desempenho pois gera um "overhead" máximo causado pelos deslocamentos do braço do disco.

A tabela da figura V.3 reúne os valores obtidos nessa primeira etapa.

Nº DE USUÁRIOS	DISTÂNCIA ENTRE ARQUIVOS	FASES							
		LEITURA				ESCRITA			
		1 (LS1)	2 (LS2)	3 (LR1)	4 (LR2)	5 (ES1)	6 (ES2)	7 (ER1)	8 (ER2)
1	–	0,15	0,73	0,53	3,16	1,18	1,83	1,73	4,11
2	130K	1,41	2,07	5,69	8,43	6,52	7,12	7,64	10,08
	1M	3,94	4,89	15,73	19,19	16,77	17,58	17,75	21,06
	2M	6,33	7,29	25,87	28,66	26,21	27,25	27,77	30,53
3	130K	2,64	3,23	9,71	12,94	10,57	11,03	12,75	15,37
	1M	7,37	8,72	29,85	34,29	31,47	31,76	32,96	37,10
	2M	12,03	13,43	49,08	53,41	50,30	51,15	51,99	55,93

Obs: Medidas expressas em segundos

FIGURA V.3

i) O Efeito da Memória Cache

As colunas 1 e 3 referem-se respectivamente às leituras sequencial e randômica: a diferença de tempo entre elas corresponde portanto à economia proporcionada pela existência da memória cache.

Para este caso, uma leitura com cache obteve uma economia de 75% em relação ao tempo gasto por uma leitura sem cache. De fato, o cache evita, para esta aplicação, três leitu

ras de bloco para cada quatro requisições enviadas por um determinado usuário. A economia mantém sua proporcionalidade apesar do aumento de usuários e do afastamento dos arquivos.

No caso da escrita, as colunas 5 e 7 mostram no entanto, que a economia proporcionada pelo cache tende a diminuir frente ao tempo real gasto pela operação, à medida que crescem o número de arquivos ou o de usuários. Este fato se explica pois o ganho obtido por elemento é constante, envolvendo apenas as leituras incluídas na execução da primitiva (cada requisição envolve obrigatoriamente uma escrita que movimenta o braço do disco). Por outro lado, o tempo médio de deslocamento por usuário cresce em qualquer dos casos, reduzindo o ganho relativo.

A figura V.4 mostra as reduções percentuais obtidas em cada caso, comparando os acessos sequencial e randômico.

		REDUÇÃO DE TEMPO DEVIDO AO CACHE	
Nº DE USUÁRIOS	DISTÂNCIA ENTRE ARQUIVOS	LEITURA (COLUNAS 1 E 3)	ESCRITA (COLUNAS 5 E 7)
1	-	72%	31%
2	130K	75%	16%
	1M	75%	7%
	2M	76%	4%
3	130K	73%	9%
	1M	75%	4%
	2M	75%	3%

FIGURA V.4

ii) O efeito do acesso indireto para localização do registro.

Analogamente à análise das escritas sequencial e randômica, as colunas 1 e 2 (leitura sequencial da primeira e da segunda parte) diferem por uma economia absoluta constante devida à ausência de acessos indiretos na manipulação dos primeiros registros do arquivo. Esta economia é relativamente menor à medida que aumentam o número de arquivos ou usuários.

No caso da escrita (colunas 5 e 6), a economia absoluta é a mesma. Como os tempos envolvidos são maiores, a economia relativa é mais modesta, conforme podemos constatar na tabela comparativa da figura V.5.

		REDUÇÃO DE TEMPO DEVIDO À AUSÊNCIA DE ACESSO INDIRETO	
Nº DE USUÁRIOS	DISTÂNCIA ENTRE ARQUIVOS	LEITURA (COLUNAS 1 E 2)	ESCRITA (COLUNAS 5 E 6)
1	-	82%	36%
2	130K	58%	17%
	1M	25%	7%
	2M	16%	4%
3	130K	56%	16%
	1M	21%	6%
	2M	13%	4%

FIGURA V.5

5.3.2- Segunda Etapa

Observamos, na etapa anterior, uma acentuada degradação com o aumento do número de usuários e da distância entre os arquivos envolvidos. Este efeito resulta da combinação desfavorável de movimentos realizados pelas cabeças de disco, impostos pelo teste realizado.

Nesta etapa, a simples dessincronização dos programas de teste executados pelos diferentes usuários revelou uma melhoria da ordem de 65% em relação aos tempos anteriormente medidos. A tabela da figura V.6 reúne os novos valores.

Nº DE USUÁRIOS	DISTÂNCIA ENTRE ARQUIVOS	FASES							
		LEITURA				ESCRITA			
		1 (LS1)	2 (LS2)	3 (LR1)	4 (LR2)	5 (ES1)	6 (ES2)	7 (ER1)	8 (ER2)
2	130K	0,52	0,92	3,41	3,20	2,32	4,91	3,73	4,55
	1M	1,01	2,05	8,12	5,85	5,92	9,14	7,11	9,16
	2M	1,65	2,90	12,81	6,76	9,11	12,77	10,62	12,05
3	130K	0,90	0,92	3,72	4,33	4,57	3,44	4,19	4,84
	1M	2,02	2,17	8,66	8,55	8,91	7,37	8,61	9,06
	2M	3,36	3,08	13,95	10,52	13,31	10,82	13,08	11,09

Obs Medidas expressas em segundos

FIGURA V.6

5.3.3- Terceira Etapa

Agora procuraremos avaliar o "overhead" resultante em tornar atômica a atualização de uma região estrutural.

Como exemplo, escolheremos a operação de fechamento de arquivo por ser uma das "transactions" requisitadas com maior frequência.

O fechamento de um arquivo foi medido da ordem de 245ms. A mesma operação, realizada para efeitos de teste de forma não atômica, correspondeu a 147ms. Observa-se assim um acréscimo de cerca de 66%, empregado em: gravar a lista de intenções correspondente, desativá-la ao final das operações e realizar as gravações em memória estável.

5.4- Conclusão

O acesso sequencial e a manipulação dos primeiros registros de um arquivo oferecem, mesmo em ocasiões desfavoráveis, melhoria de desempenho.

O custo decrescente da memória principal incentiva a utilização do cache, sobretudo considerando-se que a densidade dos discos de tecnologia WINCHESTER tende a crescer mais rapidamente que a melhora nos tempos envolvidos nos movimentos de cabeça.

A escolha do número de blocos referenciados dentro do próprio nodo é uma decisão influenciada pelos objetivos típicos dos usuários da rede local. Uma comunidade trabalhando , por exemplo, com um grande número de arquivos de até 7,5K bytes, sentiria uma melhora significativa na performance do sistema.

Para diminuir os tempos de espera resultantes dos movimentos de cabeça poderia ser adotado o "algoritmo do elevador" |Hoare, 1974| numa tentativa de minimizar a frequência de mudança de sentido no movimento do braço. Assim, dentre as várias requisições de acesso ao monitor de operações em disco, será escolhida aquela que referenciar o cilindro mais próximo, obedecido o sentido vigente. Se não existe nenhum pedido neste caso, o sentido será então alterado.

O "overhead" imposto pela atomicidade do fechamento de um arquivo parece relevante à primeira vista. No entanto, este adicional é necessário quando o sistema necessita alocar novos blocos para o arquivo durante as operações de escrita realizadas. Podemos interpretar assim que o "overhead" resultante se "dilui" entre as referidas escritas sendo justificado pela segurança que traz à consistência do sistema.

Uma vez finalizados os testes de laboratório, passaremos a uma fase de avaliação de cabeças de série em instalações variadas, submetendo assim o sistema a perfis típicos de utilização.

CAPÍTULO VI

CONCLUSÕES

O oferecimento de um recurso comum aos usuários de um sistema distribuído abre sem dúvida possibilidades incontestáveis do ponto de vista de redução de custo e capacidade de expansão.

A forma de se fazer computação em um ambiente distribuído é no entanto recente. Estamos sofrendo ainda o impacto da integração homem, computador e rede. À medida que um novo recurso é introduzido no sistema, a maneira de utilizá-lo afeta o ambiente de operação: novos padrões de compatibilidade podem-se fazer necessários refletindo em alterações no software de base e no de aplicação.

O projeto do servidor de arquivos, juntamente com o aparecimento das primeiras redes locais no mercado brasileiro, vem estimular novos investimentos de "software" e "hardware" no sentido de traçar o perfil dos usuários, aproveitar facilidades, investigar vantagens e desvantagens trazidas pelo processamento distribuído e pelo compartilhamento de recursos.

O conjunto de serviços oferecidos pelo SGA corresponde a uma posição intermediária no possível espectro de funções realizadas por um sistema de arquivos convencional. Levá-lo até o seu grau máximo de complexidade envolveria acrescentar serviços tais como variedade de métodos de acesso e facilidades para manipulação de bancos de dados. Julgamos que tais encargos são, no momento, satisfatoriamente cumpridos pelos equipamentos usuários. Dispensá-los significou para nós: aproveitamento da capacidade de processamento dos usuários;simplificação do projeto do servidor; e facilidade no processo de adaptação

ao SA. O projeto adapta-se portanto a elementos de alto e médio grau de responsabilidade, representantes de uma faixa considerável dos usuários de rede local.

Sob o ponto de vista de segurança da informação, podemos dizer que o SGA cumpre as exigências mínimas: autenticação de usuários através de passwords e definição simplificada de direitos de acesso.

Por outro lado, o alto grau de compartilhamento (a nível de registros) assim como a preocupação efetiva com a consistência dos dados constituem ferramentas simples mais eficazes na manipulação de informações sobretudo quando voltadas para banco de dados.

O "overhead" imposto pela atomicidade das "transactions" não é tão relevante quanto possa parecer à primeira vista; as listas de intenções só são ativadas quando novos blocos são alocados e desalocados, refletindo-se na operação de fechamento do arquivo. Operações de criação de arquivos e alteração de atributos também são realizadas atomicamente mas ocorrem com frequência menor.

Finalmente, a flexibilidade na organização dos arquivos conferiu aos usuários a liberdade de manter estruturas de diretórios próprias para suas aplicações.

A nossa principal orientação foi no sentido de excluir o que não era absolutamente essencial, conservando o que retornasse maior benefício com implementação relativamente simples. Ganharemos assim experiência a partir de um projeto de fácil manutenção.

Deste trabalho resultou um protótipo que se encontra

atualmente em fase de elaboração, a ser industrializado e comercializado nos próximos meses.

ANEXO I

REDE LOCAL CETUS

A.1.1- Descrição Geral

A Rede Local Cetus apresenta topologia em barra serial, utilizando como método de acesso o CSMA/CD. Sua espinha dorsal é formada de equipamentos inteligentes, denominados nodos, que se interligam entre si através de um par trançado ou cabo coaxial. A cada um dos nodos se conecta um equipamento de computação, denominado equipamento usuário, que pode ser um computador de qualquer tipo ou um periférico. A topologia do sistema fica conforme a figura AI.1.

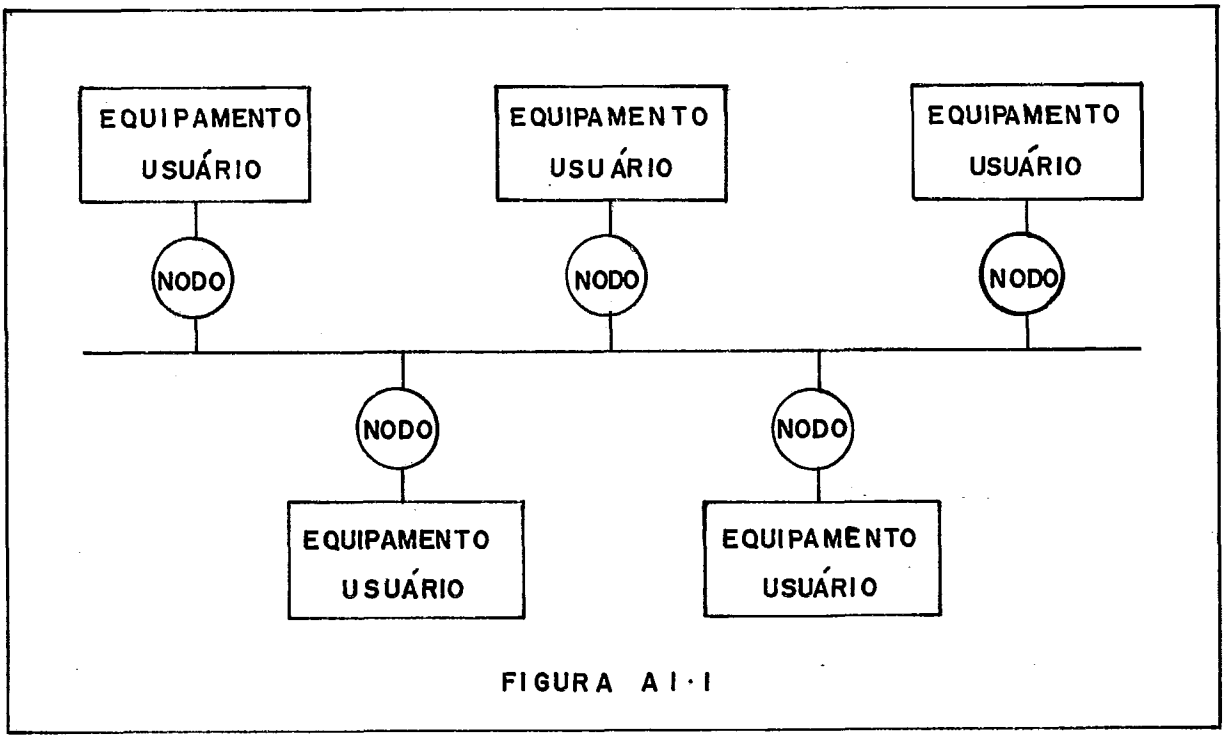


FIGURA A I·I

Cada equipamento se liga ao nodo por meio de uma interface serial (EIA RS 232C) ou paralela (CENTRONICS).

A arquitetura da rede é totalmente distribuída, não existindo controle ou supervisão centralizada. Quanto às fun-

ções de comunicação, todos os nodos são iguais e concorrem em condições de igualdade pela posse da barra serial para transmitir pacotes.

Os nodos implementam até o nível quatro (nível de transporte) do modelo de referência para interconexão de sistemas abertos, recomendado pela ISO. Realizam assim tarefas tais como: protocolos de linha, método de acesso,integridade de comunicação, endereçamento, transferência de dados, formação de pacotes, controle de fluxo, temporização, deteção e recuperação de falhas na comunicação, manutenção de circuitos virtuais.

A.1.2- Descrição do Nodo

O nodo é implementado com um microprocessador de 16 bits com memória PROM, RAM, circuitos de interface com a rede e com o equipamento usuário.

O nodo ao ser ligado, entra no estado INERTE. Neste estado ele só aceita comunicação dos outros nodos da rede ou um sinal de controle ("BREAK") do seu equipamento usuário.

Se receber um BREAK, o nodo passa para o estado de controle. Neste estado, o nodo espera do usuário comandos típicos para conectá-lo (LIGUE) ou desconectá-lo (DESLIGUE) de um outro nodo.

O nodo, uma vez conectado a um outro, passa para o estado LIGADO do qual sairá quando receber um DESLIGUE pela rede ou pelo seu equipamento usuário.

A.1.3- Resumo das Características Técnicas

i) Rede Cetus

- Meio de Transmissão: par trançado
- Topologia: barra (serial)
- Taxa de Transmissão: 1M bps
- Alcance máximo: 300m
- Capacidade máxima: 255 equipamentos, de diferentes tipos
- Protocolo de linha: SDLC
- Protocolo de acesso: CSMA-CD
- Não há elemento centralizador

ii) Nodo

- Interface entre equipamento usuário e rede
- Implementação: microprocessador INTEL (16 bits)
- Protocolos implantados nonnodo
    - . Até parte do nível 4 da ISO
    - . Transparentes ao equipamento usuário
- Conexão com o equipamento usuário
    - . Interface padrão RS 232C (comunicação serial assíncrona até 19.2K bps)
    - . Interface tipo Centronics (comunicação paralela)
- Transparência total de dados
    - . Qualquer configuração de caracter é permitida

- Procedimentos internos do nodo

    . Protocolos de rede

    . Esquema de acesso

    . Integridade da comunicação

    . Endereçamento

    . Transferência de dados

    . Empacotamento de mensagens

    . Controle de fluxo

    . Temporizações

    . Deteção e recuperação em caso de falhas

ANEXO II

UM EXEMPLO DE ADAPTAÇÃO PROTOCOLAR

A2.1- O CP/NET

O CP/NET é uma extensão do sistema operacional CP/M que permite acessar uma combinação de recursos locais e remotos. Permite assim que programas aplicativos, sem necessidade de modificação, possam compartilhar discos e impressoras remotos interligados através de uma rede local.

Para garantir independência da rede utilizada, o software do CP/NET é realizado em duas partes.

i) um módulo lógico (NDOS) encarregado de:
   - selecionar as chamadas ao sistema referentes a dispositivos remotos
   - preparar as mensagens adequadas a serem enviadas através da rede
   - reformatar o resultado recebido, devolvendo-o ao programa que realizou a chamada

ii) um módulo de adapatação (SNIOS) responsável por:
   - prever rotinas que permitam receber e transmitir mensagens através da rede (adaptação ao subsistema de transporte da rede particular adotada)
   - efetuar a adaptação protocolar com o elemento remoto que realizará o serviço requisitado (no nosso caso, o SGA)

A2.2- Compatibilidade Funcional CP/NET e SGA

Além dos serviços clássicos já fornecidos pelo CP/M (manipulação de arquivos e diretórios), o CP/NET definiu servi

ços adicionais apropriados ao novo sistema distribuído tais como controle de sessão, bloqueio de arquivos e de registros.

Neste caso particular, a correspondência existente entre as chamadas de sistema no CP/NET e as funções de serviço do SA ajuda a visualizar a compatibilidade funcional. Basta comparar as funções abaixo relacionadas com as primitivas descritas no capítulo III.

i) funções clássicas do CP/M
   - criação de arquivo
   - abertura de arquivo
   - fechamento de arquivo
   - busca em diretório
   - remoção de arquivo
   - leitura de registro (seqüencial e randômica)
   - escrita em registro (seqüencial e randômica)

ii) funções adicionadas pelo CP/NET
   - abertura de sessão
   - encerramento de sessão
   - bloqueio de registro
   - desbloqueio de registro

A2.3- Compatibilidade protocolar CP/NET e SGA

Uma vez garantida a compatibilidade funcional, um trabalho de adaptação protocolar inclui dois módulos:

- interpretador de funções - responsável por desdobrar Cada chamada de sistema em uma ou mais primitivas adequadas do SGA

- tradutor de formato - responsável por converter os parâmetros de controle de arquivo usados no sistema

local pelos parâmetros equivalentes no sistema remo
to.

No caso particular que estamos analisando, o interpre tador de funções é muito simples devido à correspondência fun - cional já observada.

O tradutor de formato por sua vez manipula principal- mente uma estrutura do CP/M denominada "FILE CONTROL BLOCK" (FCB). Isso ocorre pois as mensagens entregues pelo NDOS ao SNIOS descrevem os arquivos através do FCB. Os principais cam- pos que o constituem são:

dr: número do drive onde se encontra o arquivo

nome: nome do arquivo em ASCII

ex: número da extensão e modo de abertura

rc: número de registros alocados para a extensão

cr: registro corrente

r012: registro a ser usado em modo aleatório

O tradutor de formato deverá gerar os parâmetros ne- cessários ao SGA, a partir das informações contidas nos campos acima.

Se o usuário optar por usar o CIA do arquivo ao in- vés do seu nome completo, o SNIOS deverá manter uma tabela de correspondência entre nome e CIA.

Em caso de usar o nome completo, o campo dr corresponde ao que o SA nomeia diretório. No CP/M, o referido campo está restrito a apenas 16 valores enquanto no SA a concepção é muito mais ampla podendo representar um objeto dentro de uma estrutura multinível. Sugerimos, neste caso, a definição de um programa utilitário que mapeie os dezesseis possíveis valores do campo dr em diretórios do SA, conforme a escolha do usuário.

Os campos ex e r012 serão utilizados no cálculo do registro lógico a ser entregue ao SA. No CP/M, um arquivo é composto por diversas extensões, cada uma delas renumerando os registros a partir de zero. No SA, não existe este conceito, sendo um arquivo considerado um vetor de registros lógicos.

Os campos rc, mapa e cr são controlados pelo próprio SA.

ANEXO III

FUNÇÕES DE SERVIÇO

- ABERTURA DE SESSÃO

```
function AbreSessão (Usuário, Password : Cadeia) : E0..E2;
begin
      (* busca o nome Usuário na RAIZ (BuscaH) *)
      if (* busca sem sucesso *)
         then return (E1); (* usuário não reconhecido *)
      if (* password não coincide *)
         then return (E2); (* password inválida *)
      (* diretório corrente := diretório-raiz do Usuário *)
      if (* já existe conexão anterior (Conexão[Cv] está ati-
            va) *)
         then (* ativa rotina de encerramento de sessão *)
      (* Conexão[Cv] := entrada da RAIZ ocupada pelo Usuá-
         rio *)
      return (E0) (* sucesso *)
end;
```

- ENCERRAMENTO DE SESSÃO

```
- procedure EncerraSessão;
  begin
       if (* existe conexão (Conexão[Cv] está ativa ) *)
          then begin
               (* ativa a função de fechamento de arquivo para
                  todos os canais (entradas da TabelaCanal )
                  abertos para este circuito virtual *)
               (* desativa o diretório corrente *)
               end
  end;
```

- CRIAÇÃO DE ARQUIVO

```
function CriaArq (Arquivo : Cadeia; Tipo : Objetos;
DireitoPúblico : Direitos; TamanhoReg : 1..TamBloco;
var Canal : TipoCanal; var  Cia : TipoCia) : E0..E7;
begin
     (* localiza, através de uma busca (BuscaV), o diretório
        no qual se deseja criar o arquivo . (diretório ances
        tral) *)
     if (* busca sem sucesso *)
        then return (E1); (* diretório ancestral  inexisten -
                             te *)
     if (* objeto encontrado é do tipo não-diretório *)
        then return (E1); (* diretório ancestral  inexisten-
                             te *)
     if (* usuário não tem direito de escrita sobre o diretó-
           rio ancestral *)
        then return (E2); (*direito de acesso inválido *)
     (* busca o nome de referência do arquivo no diretório an
        cestral (BuscaH) *)
     if (* busca sem sucesso *)
        then begin (* o arquivo a ser criado não existe *)
             (* busca nodo livre *)
             if (* busca sem sucesso *)
                then return (E5); (* número máximo de arqui -
                                     vos excedido *)
             (* grava nova entrada no diretório ancestral com
                as características especificadas como parâme-
                tros (GravaNovaEntrada) *)
             if (* não há mais entradas disponíveis no diretó
                   rio ancestral *)
```

```
                then return (E7); (* número máximo de entra-
                                     das excedido *)
            end
        else begin (* o arquivo já existia *)
            if (* o antigo arquivo era tipo diretório *)
                then return (E3); (* arquivo tipo diretó -
                                     rio *)
            (* verifica se o arquivo já estava aberto, con-
               sultando a TabelaAtiva *)
            if (* arquivo se encontra aberto *)
               then return (E4); (* arquivo já aberto *)
            (* reinicializa o arquivo (ReinicializaEntra
               da) *)
            end;
    (* procura entrada livre na TabelaCanal *)
    if (* busca sem sucesso *)
       then return (E6); (* não há canais disponíveis *)
    (* busca uma entrada livre na TabelaAtiva e preenche os
       campos adequados com os dados do arquivo e modo de
       abertura EsclusivoNormal e Contador =1 *)
    (* preenche os campos da entrada da TabelaCanal com o
       apontador para a entrada da TabelaAtiva e
       RegCorrente =0 *)
    (* Canal := entrada da TabelaCanal *)
    (* Cia := CIA do arquivo *)
    return (E0) (* sucesso *)
end;
```

- CRIAÇÃO DE DIRETÓRIO

```
function CriaDir (*Diretório : Cadeia;
DireitoPúblico : Direitos; var Canal : Tipo Canal) : E0..E5;

begin
     (* localiza, através de uma busca (BuscaV), o diretório
        no qual se deseja criar a nova entrada (diretório an
        cestral) *)
     if (* busca sem sucesso *)
        then return (E1); (* diretório ancestral inexistente *)
     if (* usuário não tem direito de escrita sobre o diretó-
           rio ancestral *)
        then return (E3); (* direito de acesso inválido *)
     (* busca o nome de referência do diretório no diretório
        ancestral (BuscaH) *)
     if (* busca sem sucesso *)
        then begin (* o arquivo a ser criado não existia *)
             (* busca nodo livre *)
             if (* busca sem sucesso *)
                then return (E4); (* número máximo de arqui-
                                     vos excedido *)
             (* grava nova entrada no diretório ancestral com
                as características especificadas (GravaNo-
                vaEntrada)*)
             if (* não há mais entradas disponíveis no diretó
                   rio ancestral *)
                then return (E5); (* número máximo de entra -
                                     das excedido *)
             return E0 (* sucesso *)
             end
        else return  (E2) (* arquivo já existente *)
end;
```

- BUSCA INICIAL EM DIRETÓRIO

```
function BuscaInicial (Cia : TipoCia; Referência : Cadeia;
var CiaArq : TipoCia; var Nodo : TipoNodo) :E0..E3;

begin
     CiaDirBuscaCorrente[Cv]   := Cia;
     ReferênciaBuscaCorrente[Cv] := Referência;
     EntradaBuscaCorrente[Cv]  := 0;
     return Busca (CiaArq, Nodo) (* realiza uma busca a par -
                                    tir da primeira  entrada
                                    do novo diretório de bus-
                                    ca corrente *)

end;
```

- BUSCA EM DIRETÓRIO

```
function Busca (var CiaArq : TipoCia;
var Referência : Cadeia; var Nodo : TipoNodo *) : E0..E3;

begin
     if (* o nodo correspondente ao CIA do diretório de busca
           corrente está livre *)
        then return (E1); (* não existe *)
     if (* o arquivo especificado para a busca não é tipo di-
           retório ou diretório-raiz *)
        then return (E1);
     if (* o usuário não tem nenhum direito de acesso  sobre
           o diretório de busca corrente *)
        then return (E2); (*direito de acesso inválido *)
     (* realiza busca sobre o diretório de busca     corrente
        (BuscaH) *)
     if (* busca sem sucesso *)
        then return (E3) (* não foi encontrado *)
```

```
        EntradaBuscaCorrente [Cv] := EntradaBuscaCorrente [Cv]+1;
        (* carrega a variável Nodo com o conteúdo do nodo cor-
           respondente ao arquivo encontrado *)
        (* Referência := nome de referência do arquivo encontrado *)
        return (E0); (* sucesso *)
    end;
```

- REMOÇÃO DE ARQUIVO (OU DIRETÓRIO)

```
    function Remove (Arquivo : Cadeia) : E0..E5;
    begin
        (* localiza, através de uma busca (BuscaV), o arquivo
           que se deseja remover *)
        if (* busca sem sucesso *)
           then return (E1); (* arquivo inexistente *)
        if (* usuário não tem direito de escrita sobre o diretó-
              rio ancestral ou se o mesmo é diretório-raiz *)
           then return (E2); (* direito de acesso inválido *)
        if (* usuário não tem direito de escrita sobre o arquivo *)
           then return (E2); (* direito de acesso inválido *)
        if (* arquivo a ser removido é do tipo diretório *)
           then begin
                (* realiza uma busca sobre todas as entradas
                   do arquivo (BuscaH) *)
                if (* existe alguma entrada *)
                   then return (E3); (* diretório contém entradas *)
                if ( * o arquivo é diretório corrente para algu-
                       ma conexão *)
                   then return (E4); (* tipo diretório corrente *)
                end
           else begin (* tipo não diretório *)
                if (* o arquivo já estava aberto *)
                   then return (E5); (* arquivo já aberto *)
```

```
            end;
      (* remove a entrada do diretório ancestral, liberando o
         nodo envolvido e os blocos alocados (RemoveEntrada) *)
      return (E0) (* sucesso *)
end;
```

- ALTERAÇÃO DE NOME DE ARQUIVO (OU DIRETÓRIO)

```
function (Arquivo, NovoNome : Cadeia) :E0..E5;
begin
      (* localiza, através de uma busca (BuscaV), o arquivo
         cujo nome se deseja alterar *)
      if (* busca sem sucesso *)
         then return (E1); (* objeto inexistente *)
      if (* usuário não tem direito de escrita sobre o arquivo *)
         then return (E2); (* direito de acesso inválido *)
      if (* arquivo é do tipo diretório *)
         then begin
              if (* é diretório corrente para alguma conexão *)
                 then return (E3) (* diretório corrente *)
              end
         else begin
              if (* o arquivo já estava aberto *)
                 then return (E4) (* arquivo já aberto *)
              end
      (* busca o novo nome de referência no diretório ancestral *)
      if (* busca com sucesso *)
         then return (E5); (* nome já existente *)
      (* grava referência na entrada do diretório ancestral
         correspondente ao arquivo (GravaReferência) *)
      return (E0) (* sucesso *)
end;
```

- ALTERAÇÃO DE DIREITO DE ACESSO

```
function AlteraDireito (Cia : TipoCia ; DireitoPróprio,
DireitoPúblico : Direitos) :E0..E4;

begin
     if (* o nodo correspondente ao Cia está livre *)
        then return (E1); (*arquivo inexistente *)
     if (* o usuário não é dono do arquivo *)
        then return (E2); (* direito de acesso inválido *)
     if (* arquivo é do tipo diretório *)
        then begin
             if (* é diretório corrente para alguma conexão *)
                then return (E4) (* diretório corrente *)
             end
        else begin
             if (* o arquivo já estava aberto *)
                then return (E3); (* arquivo já aberto *)
             end;
     (* grava os novos direitos de acesso do arquivo (Gra-
        vaDireitos) *)
     return (E0) (* sucesso *)
end;
```

- ALTERAÇÃO DE PASSWORD

```
procedure AlteraPass (Password : Cadeia);

begin
     (* altera a password do usuáio que ocupa este circuito
        virtual (GravaPassword) *)
end;
```

- ALTERAÇÃO DE DIRETÓRIO CORRENTE

```
function AlteraDirCorrente (Cia : TipoCia) :E0..E2;

begin
      if (* o nodo correspondente ao CIA está livre *)
         then return (E1); (*diretório inexistente *)
      if (* o usuário não tem direitonenhum sobre o    arquivo
            especificado pelo CIA *)
         then return (E2); (* direito de acesso inválido *)
      CiaDirCorrente := Cia;
      return (E0)
end;
```

- BLOQUEIO DE REGISTRO

```
function Bloqueio (Canal : TipoCanal;
Registro : TipoRegistro; NumReg : O..  NumRegBloqMax)  : E0..E4;

begin
      if (* canal não está aberto *)
         then return (E1); (* canal não aberto *)
      if (* modo de abertura é diferente   de Compartilhado *)
         then return (E2); (* modo incompatível *)
      if (* pelo menos um dos registros a serem bloqueados es-
            tá fora do arquivo *)
            then return (E4); (* registro fora do arquivo *)
      if (* pelo menos um dos registros a serem bloqueados  já
            se encontra bloqueado por outro canal *)
            then return (E3); (* já bloqueado *)
       TabelaCanal [Canal]. RegBloq := Registro;
      TabelaCanal[Canal]. NumRegBloq := NumReg;
      return (E0) (* sucesso *)
end;
```

- DESBLOQUEIO DE REGISTRO

```
function Desbloqueio (Canal : TipoCanal) :E0..E2;

begin
      if (* canal não está aberto *)
         then return(E1); (* canal não aberto *)
      if (* modo de abertura é diferente de Compartilhado *)
         then return (E2); (* modo incompatível *)
      TabelaCanal[Canal]. NumRegBloq := 0;
      return (E0)
end;
```

BIBLIOGRAFIA

1. BIRRELL, A.D.; NEEDHAM, R.M., "A Universal File Server", IEEE Transactions on Software Engineering, 6.5, 1980.

2. BOCHMANN, G., "Concepts for Distributed Systems Design", Springer - Verlag, 1983.

3. DAY, J.D.; "Resource Sharing Protocols", IEEE Computer, 1979.

4. DELLAR, C.N.R., "A File Server of a Network of Low Cost Personal Microcomputers", Software-Practice and Experience, 12, 1051-1068, 1982.

5. DIGITAL RESEARCH, "CP/M Interface Guide", 1979.

6. DIGITAL RESEARCH, "CP/NET Network Operating System Reference Manual", 1982.

7. DION, J., "The Cambridge File Server", Operating Systems Review, 14, 4, 26-35, 1980.

8. DONOVAN, J.J., "Systems Programming", Mc Graw-Hill, 1972.

9. GAIT, J.; "The Password Encryption Problem", Operating Systems Review, 12, 1, 1978.

10. GIEN, M., "A File Transfer Protocol", Proc. Computer Network Protocols Symposium, 1978.

11. IEEE PROJECT 802, "Local Area Network Standards", 1982.

12. METCALFE, R.; BOGGS, D., "Ethernet: Distributed Pachet Switching for Local Computer Networks",Communications of the ACM, 19, 7, 1976.

13. MITCHELL, J.G.; DION, J.; "A Comparison of Two Network-Based File Servers", Proceedings of the Eighth Symposium on Operating System Principles, 1981.

14. ORGANICK, E.I., "The Multics System", MIT Press, 1972.

15. OSI REFERENCE MODEL, IEEE.Trans Communications, 1980.

16. ROBERTSON, G.; THOMPSON, M.; ACCETTA, M.; "The Design of Network-Based Central File System", Technical Report, Carnegie-Mellon University, 1980.

17. ROWE, L.A.; BIRMAN, K.P., "A Local Network Based on the UNIX Operating System", IEEE Transactions on Software Engineering, 8, 2, 137-146, 1982.

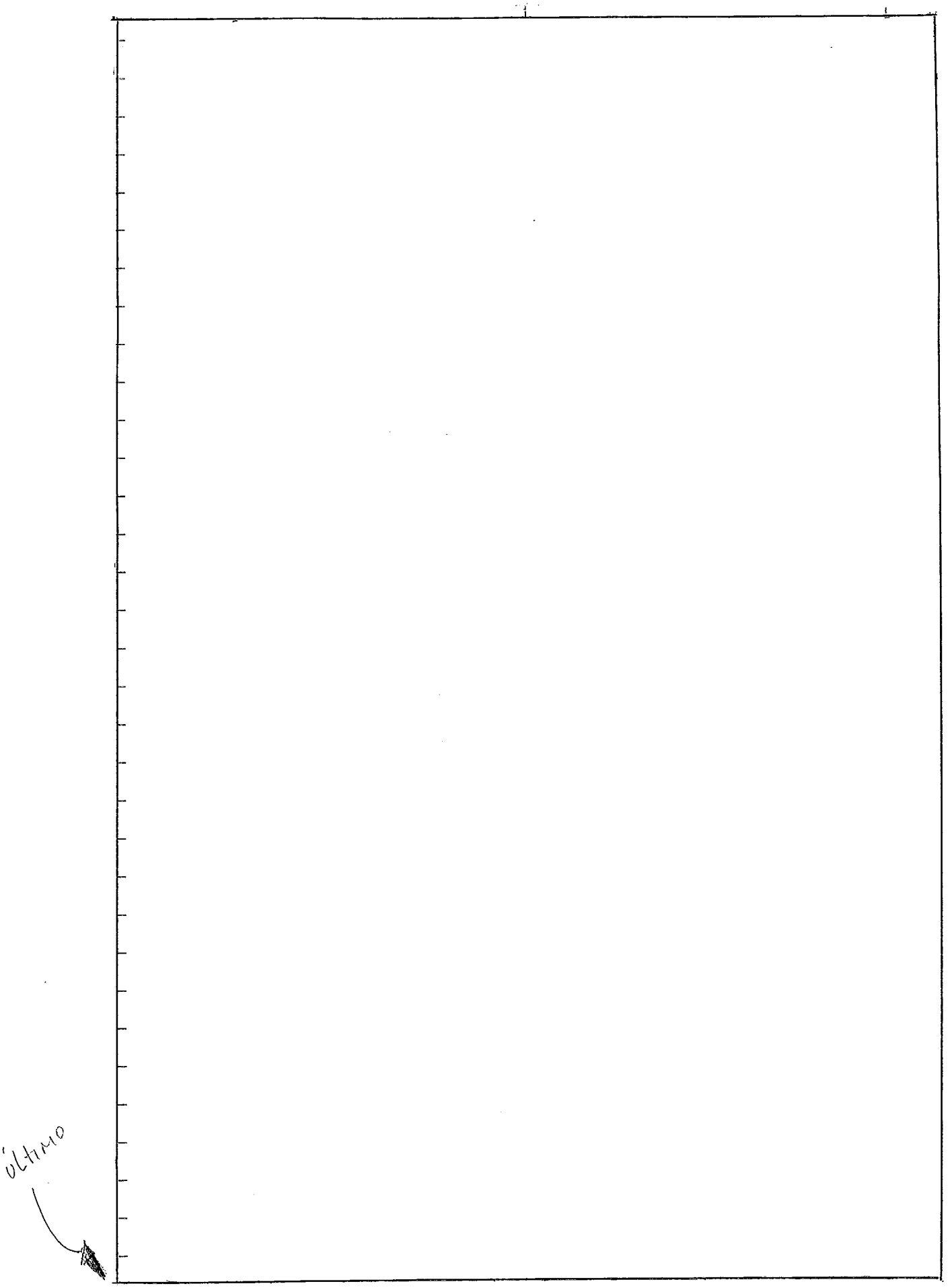
18. SCHIMTZ, E.A.; GRACIA, J.; SOUZA, L.C.; MONTEIRO, M.C.B., "Um Servidor de Arquivos para uma Rede Local de Computadores", XVI Congresso Nacional de Informática,1983.

19. SOUZA, M.L.D., "Um Servidor de Discos em um Ambiente de Processamento Distribuído", Projeto de Fim de Curso, 1983.

20. STURGIS, H.E.; MITCHELL, J.G.; ISRAEL, J., "Issues in the Design and Use of a Distributed File System", Operating Systems Review, 14, 3, 55-69, 1980.

21. SWINEHART, D.; MCDANIEL, G.; BOGGS, D.; "WFS: A Simple Shared File System for a Distributed Environment", Proceedings of the Seventh Symposium on Operating Systems Principles, 1979.

22. THOMPSON, K.; RITCHIE, D.M.; "The UNIX Time-Sharing System", Bell Laboratones, 1974.


ÚLTIMO

23. THOMPSON, K.,, "UNIX Implementation", Bell Systems Tech., 1978.

24. TSICHRITZIS, D.C; BERNSTEIN, P.A., "Operating Systems", Academic Press, 1974.

25 WALDEN, D.C., "The Evolution of Host-to-Host Protocol Technology", IEEE Computer, 1979.

26. WATSON, R.W., "Timesharing System Design Concepts", Computer Science Series, 1970.

27. WECHER, S., "Computer Network Architectures",IEEE Computer, 1979.

28. WILKES, M.V.; NEEDHAM, R.M., "The Cambridge Model Distributed System", Operating Systems Review, 14, 21-29, 1980.

29. HOARE, C.A.R., "Monitors: An Operating System Structuring Concept", Programming Methodology, Springer - Verlag, 1974.