DESENVOLVIMENTO DE UM SERVIDOR DE ARQUIVO PARA CLIENTES DE BAIXO CUSTO

Raimundo José de Araújo Macêdo

UNIVERSIDADE ESTADUAL DE CAMPINAS

INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO

CAMPINAS - SÃO PAULO

BRASIL

DESENVOLVIMENTO DE UM SERVIDOR DE ARQUIVO

PARA CLIENTES DE BAIXO CUSTO

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Raimundo José de Araújo Macêdo e aprovada pela comissão julgadora.

Campinas, 21 de agosto de 1986.

Prof. Dr. Célio Cardoso Guimarães

Orientador

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação.

Agradecimentos

Ao Prof. Dr. Célio C. Guimarães pela orientação durante o período deste trabalho.

A Joselita Filha pelos desenhos.

Ao Centro de Processamento de Dados da Universidade Federal da Bahia, através de seu diretor, Geovane Cayres Magalhães, pelo apoio e incentivo.

Finalmente, aos colegas do CPD-UFBa pelo estímulo.

Sumário

Neste trabalho é proposto um Servidor de Arquivo que executa todas as funções de manipulação de arquivos existentes em um sistema operacional convencional (CP/M) e algumas funções adicionais, como compartilhamento de arquivos. O protótipo desenvolvido consta do software do Servidor de Arquivo, do sistema operacional da Estação de Trabalho adaptado e da comunicação entre o Servidor de Arquivo e a Estação de Trabalho.

INDICE

1.0 INTRODUÇAO

Um **Servidor de Arquivo (SA)** é um computador que, através de uma rede (geralmente local), é utilizado como um sistema de arquivos compartilhado.

Com o crescente uso de sistemas distribuídos, a utilização de Servidores de Arquivo tem se tornado necessária tanto por razões econômicas, as quais são justificadas pelo compartilhamento de recursos caros (ex: discos rígidos), quanto por razões práticas, para viabilizar certos tipos de aplicação, tais como compartilhamento de arquivos.

O objetivo central deste trabalho é o desenvolvimento de um Servidor de Arquivo destinado à clientes de **baixo custo**, tipicamente micro-computadores de 8 bits.

Neste trabalho é proposto um SA que executa todas as funções de manipulação de arquivos existentes em um sistema operacional convencional (CP/M) e algumas funções adicionais, como compartilhamento de arquivos entre nós da rede.

Na presente abordagem explora-se a popularidade do sistema operacional CP/M (Control Program / Monitor) para implementar um Servidor de Arquivo que é uma extensão das funções de arquivos do CP/M, possibilitando a utilização do SA de forma transparente ao programa de aplicação.

O protótipo desenvolvido consta do software do SA, do sistema operacional da Estação de Trabalho (ET) adaptado e da comunicação SA/ET.

O capítulo 2.0 é uma abordagem geral sobre Servidores de Ar-

quivo. Procura-se dar ênfase às características de Servidores atualmente implementados cujas descrições são disponíves. No capítulo 3.0 é descrito o protótipo implementado para o sistema proposto e no capítulo 4.0 são apresentadas sugestões para extensões ao protótipo.

2.0 - SERVIDORES DE ARQUIVO

2.1 - Descrição do Ambiente

2.1.1 - Conceitos Básicos

O termo **Servidor de Arquivo** tem sido usado geralmente para significar um sistema que é disponível numa rede (geralmente local), cuja principal finalidade é o armazenamento e recuperação de grandes volumes de dados.

Embora o termo **SA** sugira um sistema de arquivos com capacidade de gerenciar diretórios de arquivos, impor diferentes restrições a grupos de usuários distintos, identificar arquivos através de seus nomes,etc., observa-se que muitas vezes é utilizado para denominar sistemas (ex: **"Woodstock File Server"** [SWINEHART 79]) que não oferecem praticamente nenhuma das facilidades encontradas em sistemas de arquivos convencionais.

Servidores de Arquivo também têm sido usados de uma forma distribuída, onde vários servidores comunicam-se entre si cooperando a fim de permitir que um cliente "ao mesmo tempo" tenha acesso a arquivos em servidores distintos, como é o caso do **"Xerox Distributed File System"** [STURGIS 80].

Além da utilização em redes locais, Servidores de Arquivo estão atualmente implementados em redes tipo "internetwork" onde várias redes locais estão interligadas. Num ambiente desse, o serviço de arquivo mais atraente seria aquele que suportasse vários servidores, de modo a ter-se a ilusão de um único serviço de arquivo distribuído.

Mitchell [MICTHELL 83] definiu um **Serviço de Arquivo** como

sendo um software "rodando" em uma ou mais máquinas, um **Servidor de Arquivo** como sendo o software do Serviço de Arquivo "rodando" em uma máquina e um **CLIENTE** de um Serviço de Arquivo como um software que pode ou não estar diretamente operando em benefício de um usuário humano.

Usaremos neste texto os termos acima mencionados de acordo com a definição de Mitchell.

A figura 1 esclarece o quadro figurado por Mitchell no que se refere as camadas que envolvem o **CLIENTE**.

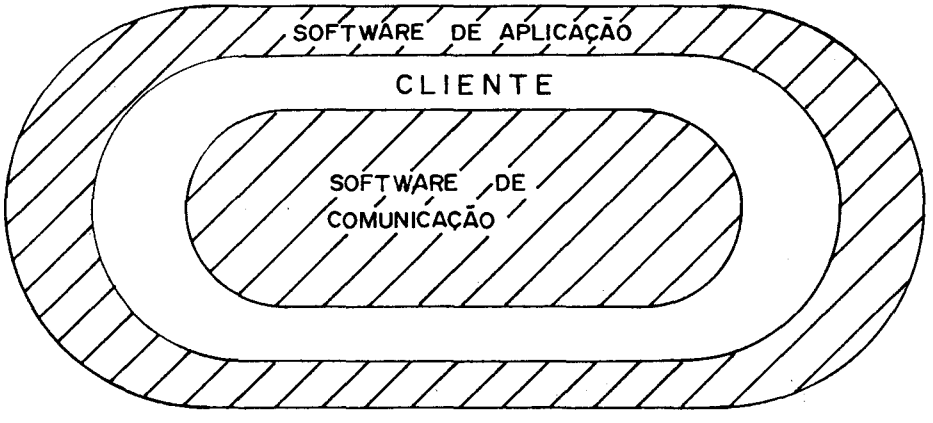

Figura 1 - Camadas de Software que envolvem o **Cliente**.

A essa altura, um ponto importante a ser abordado é a interface entre o **Cliente** e o **Software de Comunicação**, a qual é comumente referida como **INTERFACE DO CLIENTE**. Essa interface representa o conjunto de operações através da qual se tem acesso ao SA. Essas operações são em geral primitivas (ex: aloque N unida-

des de armazenamento a partir do endereço X). Assim, o papel do **Cliente** é formar, por sua vez, uma interface mais adequada à utilização por parte do **Software de Aplicação**.

2.1.2 - COMUNICAÇÃO CLIENTE/SERVIDOR DE ARQUIVO

Clientes e servidores de arquivo se comunicam através da troca de mensagens um para o outro. Se um **CLIENTE** deseja, por exemplo, ter acesso a um arquivo num **SA** , ele manda uma mensagem com um **PEDIDO**, especificando o tipo e os parâmetros necessários. Em seguida, ele aguarda que o **SA** responda através de uma mensagem **RESPOSTA** àquele pedido.

2.2 - PARAMETROS CONSIDERADOS NO PROJETO DE UM SERVIDOR DE ARQUIVO

2.2.1 - Universalidade do Servidor de Arquivo

Como já mencionado anteriormente, SERVIDORES DE ARQUIVO têm sido usados para desde transferência convencional de arquivos até como base para sistema de memória virtual.

As implementações variam entre dois extremos:

Mais simples: O SA comporta-se para seus clientes como um **DISCO REMOTO**, cabendo ao cliente a responsabilidade sobre qualquer estrutura que ele deseje para seus dados;

Mais sofisticado: O SA comporta-se como um sistema de arquivos completo à semelhança dos encontrados em sistemas operacionais do tipo "time-sharing".

A escolha entre esses dois extremos vai depender de quais mecanismos de um sistema de aquivo o projetista deseja implementar no SERVIDOR DE ARQUIVO e, consequentemente, quais mecanismos ele deixa para serem implementados pelo cliente.

Escolhendo-se ter um SA comportando-se como um disco remoto simplifica-se o projeto e implementação, em contra partida, o software dos clientes gastariam bastante recursos computacionais para implementar estruturas de alto nível, encontradas em sistemas de arquivo convencionais. A outra opção é bastante cômoda no que diz respeito à recursos computacionais do cliente, pois com um SA possuindo todas as funções de alto nível de um sistema

de arquivo, o "código" necessário à implementação do cliente seria bastante reduzido. A desvantagem aqui seria a de distintos clientes terem de usar um mesmo modelo de sistema de arquivo.

Num ambiente de propósito geral (rede), diferentes clientes podem querer usar o servidor de arquivo de formas diversas, atendendo a diferentes necessidades; por exemplo, clientes distintos podem estar implementados em computadores diferentes, cada qual usando o seu sistema de arquivo particular.

No artigo **"A UNIVERSAL FILE SERVER"** [BIRRELL 80] Birrell e Needham sugerem a construção de um SA **UNIVERSAL**, o qual se situaria entre os dois extremos de implementação discutidos. O SERVIDOR DE ARQUIVO implementaria todas as funções existentes em um disco remoto mais as funções de gerenciamento de memória de massa que fossem consideradas comuns a sistemas de arquivo. Funções específicas de nível mais alto seriam implementadas pelos sistemas de arquivo(clientes) que as desejassem. Um exemplo de tal função de alto nível seria o mapeamento de nomes de arquivos em nomes internos reconhecidos pelo SA. Para fazer tal mapeamento, o sistema de arquivo deve se utilizar de diretórios de arquivos impondo restrições a diferentes usários .

Nessa proposta, o SA irá se responsabilizar pela manutenção de um conjunto de objetos no sistema de discos, cada objeto teria um **NOME INTERNO** e todas as operações feitas no SA seriam feitas através desse nome interno. Para que o gerenciamento desses **NOMES INTERNOS** seja feito de modo a possibilitar os vários clientes implementarem estruturas as mais diversas (sistemas de arquivos), Birrell e Needham propõem que o servidor de arquivo possua um

"SISTEMA UNIVERSAL DE ÍNDICES". Na estrutura proposta existem dois tipos de objetos:

ÍNDICE: Esse objeto contém um comjunto de nomes internos de outros objetos, somente é alterado pelo SA;

SEGMENTO: Esse objeto contém os dados do cliente.

O SA garante a existência de objetos cujos nomes internos apareçam em pelo menos um índice.

Existe um ÍNDICE MESTRE cuja existência é de responsabilidade do servidor de arquivo.

Essa estrutura forma um GRAFO ORIENTADO podendo conter CICLOS. O espaço ocupado por um objeto somente seria liberado pelo SA, quando tal objeto fizesse parte de uma componente disconexa do grafo que contém o índice mestre. Ou seja, um objeto passa a não existir se a partir do indice mestre não encontramos um caminho até ele.

Nessa proposta os mecanismos do SA (INTERFACE DO CLIENTE) que poderiam ser usados por sistemas de arquivo e outros clientes específicos, incluiriam:

1) Crie um OBJETO de um dado tipo e tamanho e quarde seu NOME INTERNO no deslocamento N dentro do ÍNDICE x;

2) Troque o tamanho do OBJETO x por N unidades;

3) Guarde o **NOME INTERNO** do **OBJETO** x dentro do **ÍNDICE** y no deslocamento N;

4) Remova o conteúdo (**NOME INTERNO**) do deslocamento N no **ÍNDICE** x;

5) "Leia" o conteúdo de N unidades começando no deslocamento P do **OBJETO** x;

6) "Escreva" N unidades a partir de um "buffer" dentro da estrutura de armazenamento começando no deslocameno P no **ÍNDICE** x.

Com essa **INTERFACE DO CLIENTE** cada cliente poderia implementar seu próprio sistema de arquivo com a raiz de seu sistema alocada dentro do **ÍNDICE MESTRE** ou acessado a partir dele. A figura 2 ilustra a implementação de um sistema de arquivo com dois níveis.

O mecanismo de **CONTROLE DE ACESSO** sugerido por Birrell e Needham para ser usado no Servidor de Arquivos seria o uso de "**CAPABILITIES**" [SALTZER 75], onde os **NOMES INTERNOS** seriam constituídos a partir de identificadores únicos.

Com o SA estruturado da forma acima descrita, torna-se possível a utilização simultânea deste, por clientes que implementam sistemas de arquivo compartilhados, por clientes que implementam sistemas de arquivo isolados e também por clientes que usam o S.A à semelhança de um **DISCO REMOTO**.

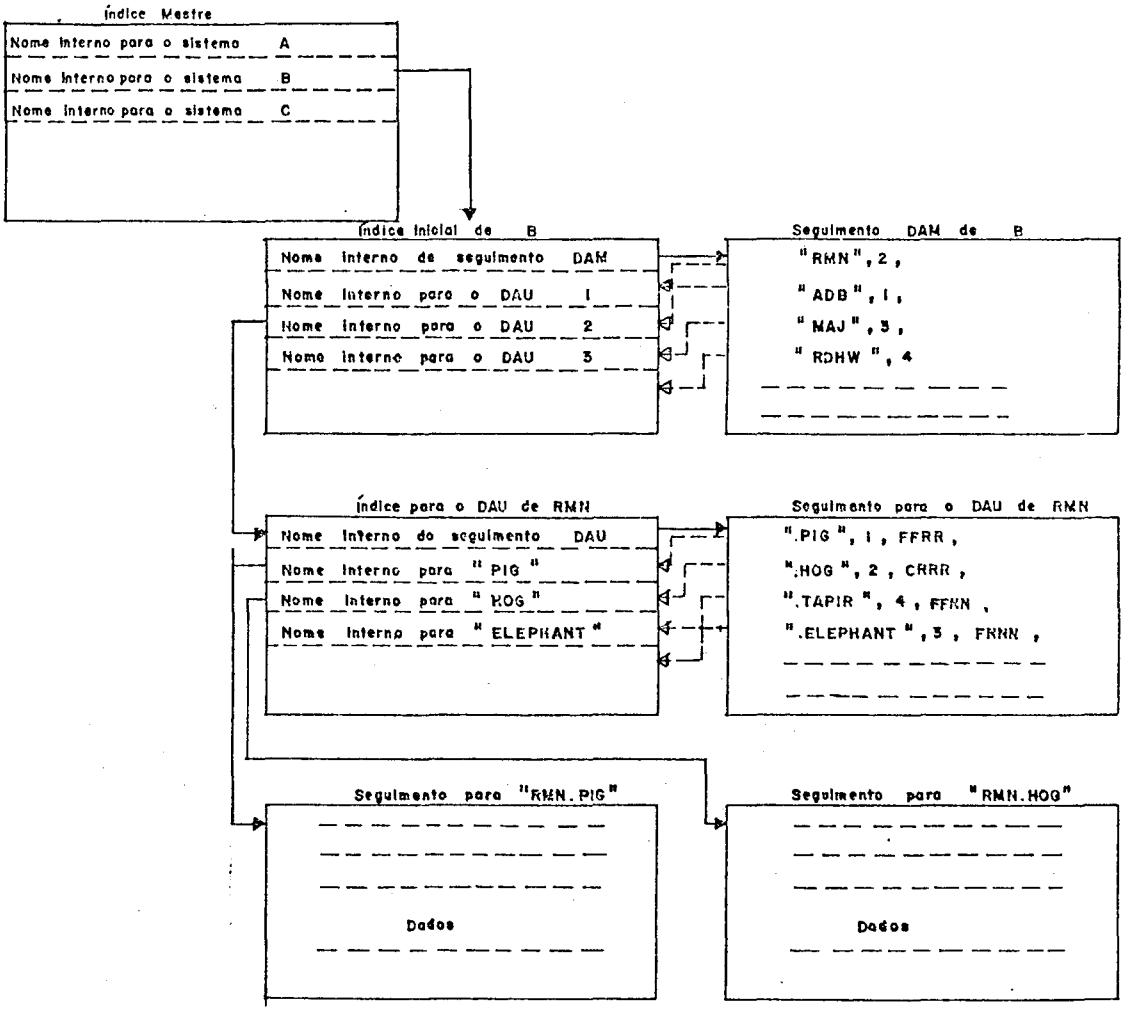

Figura 2 - Sistema de Arquivos com dois níveis [Birrell 80].

2.2.2 - Transações Atômicas

2.2.2.1 - Fundamentos

Chamaremos de **TRANSAÇÃO** a uma sequência de operações de leitura e escrita em arquivos. Mesmo com operações de leitura e escrita absolutamente confiáveis, a ocorrência de falhas de software ou de hardware (no cliente, no SA ou no meio de comunicação) no decorrer de uma transação, pode comprometer a integridade dos dados no SA. A execução de transações concorrentes também pode resultar em dados inconsistentes, pois cada operação dentro de uma transação "enxerga" os dados num estado que corresponde ao momento do início da transação com alterações efetuadas somente por essa transação. Se, por exemplo, tivéssemos duas transações, A e B, compartilhando uma conta bancária Y e a transação A fazendo transferência de fundos da conta X para Y enquanto a transação B depositando 5.000,00 em Y. E, num determinado momento, B obtivesse o saldo de Y a fim de acrescentar os 5.000,00, e logo depois parasse, sendo a transação A totalmente executada para somente assim B prosseguir. A execução dessas duas transações concorrentes feitas dessa forma ignoraria a transferência de fundos de X para Y.

Com o uso de **TRANSAÇÕES ATÔMICAS** (T.A.), a consistência dos dados seria mantida, tanto na ocorrência de falhas quanto na execução de transações concorrentes, visto que uma transação somente é atômica se ela possui duas propriedades básicas:

1) **ATÔMICA** com respeito ao acesso concorrente: Do ponto de

vista da consistência dos dados, **TRANSAÇÕES ATOMICAS** concorrentes são executadas sequencialmente, uma de cada vez;

2) **ATOMICA** com respeito à falhas: Para cada **TRANSAÇÃO ATÔMICA**, ou todas as trocas feitas (escritas) em arquivos acontecem, ou então nenhuma delas acontece.

Em algumas implementações de **TRANSAÇÕES ATÔMICAS** existentes nos servidores de arquivos atuais, em uma única transação pode-se atualizar no máximo um arquivo (ex: **"Cambridge File Server"** [DION 80]), em outras, vários arquivos (ex: **"Felix File Server"** [FRIDRICH 80]) ou em alguns casos vários arquivos e vários servidores de arquivo podem conjuntamente participar de uma única transação(ex:**"Xerox Distributed File System"** [STURGIS 80]).Alguns servidores como o **"Woodstock File Server"** [SWINEHART 79] e o **"Acorn File Server"** [DELLAR 82] não implementam nenhum mecanismo de **Transação Atômica**. Nesses casos, o trabalho de implementação de transações atômicas fica a cargo do cliente. **Paxton** [PAXTON 79] propôs um sistema de transações atômicas implementado exclusivamente pelos clientes.

2.2.2.2 - Operação

Nos servidores que restringem a atualização atômica a um único arquivo (ex: "**Cambridge File Server**") , o início de uma **Transação Atômica** é determinada como efeito colateral de um comando de abertura de arquivo (**OPEN**) e o final como efeito colateral de um comando de fechamento de arquivo (**CLOSE**). Em outros servidores, com capacidade de atualizar atômicamente vários arquivos numa transação, geralmente existem comandos específicos para manipulação de transações que incluem o início da transação, o final (commit), e também comandos para aborto de transações.

Um ponto importante na operação de transação atômica é o fechamento (**COMMIT**). Somente quando o Servidor de Arquivo confirma ter fechado (**COMMITED**) a transação, é que podemos ter certeza que as trocas vão ser realmente efetuadas. Um ponto delicado aqui é quando o SA "cai" ou o sistema de comunicação falha logo após o cliente ter pedido para encerrar (commit) a transação e antes de receber a confirmação do SA de que a transação foi fechada (commited). Em alguns sistemas (ex:"**Xerox Distributed File System**") existe um histórico de um determinado número de transações anteriormente executadas. Esse histórico pode ser consultado pelo cliente para verificar o estado de uma transação anteriormente executada. Em servidores de arquivo que não possuem esse mecanismo, recomenda-se por exemplo, que o cliente crie em arquivo vazio dentro da transação. Logo depois que o cliente conseguir "contato" com o servidor, a existência do arquivo aberto atesta que a transação foi realmente fechada (**COMMITED**).

Se considerarmos o fato de que falhas no SA, no meio de comunicação ou no cliente não são muito frequentes e sabendo-se que, em algumas aplicações como, por exemplo, a compilação de um programa, a reexecução de uma computação é trivial, então chegaremos a conclusão de que o "overhead" causado pela implementação de Transações Atômicas nem sempre é necessário. Baseados nesses argumentos, os projetistas do **"Cambridge File Server"** se utilizaram do conceito de arquivos **especiais** e **normais**, onde somente arquivos especiais, que são declarados no início de uma transação (comando **OPEN**), são atualizados atomicamente.

A seguir são descritas sumariamente as operações (**INTERFACE DO CLIENTE**) sobre transações atômicas em três servidores de arquivo atualmente implementados. Esses três conjuntos são bastante representativos, desde que os três Servidores implementam transações atômicas de forma distintas:

1) **CFS ("Cambridge File Server" [DION 80])**

No CFS uma transação atômica é limitada somente a um objeto (ARQUIVO OU INDICE). Vários clientes podem participar conjuntamente da mesma transação, basta que o cliente que iniciou a transação passe a "capability" (indentificador do objeto) para os outros clientes participantes.

Uma transação atômica é iniciada pelo comando:

OPEN[identificador_do_objeto]

O SA verifica a existência do objeto e devolve para o cliente um identificador de transação (**IDENT_TRANS**) que será usado pelo cliente para operações sobre esse objeto. Ex:

READ[ident_trans,deslocamento,tamanho]

Essa operação devolve dados lidos a partir do arquivo identificado por ident_trans.

Para fechar (**COMMIT**) uma transação o cliente usa o comando:

CLOSE[ident_trans]

Opcionalmente o cliente pode **ABORTAR** a transação passando o parâmetro adequado junto ao identificador de transação (ident_trans). A operação CLOSE devolve o resultado da tentativa de fechar (**COMMIT**) a transação, que pode ser FECHADA (**COMMITED**) ou ABORTADA. A transação é abortada se acontecer alguma falha no servidor, ou mesmo por causa de conflitos de trancas (**LOCKS** - vide 3.3).

2) **"FELIX FILE SERVER"** - [FRIDRICH 81]

O SA FELIX manipula um conjunto de arquivos em uma única transação atômica. Identificadores de arquivos e identificadores de transação são "capabilities" [SALTZER 75], de modo que vários clientes podem participar de uma transação atômica, basta

que possuam a "capability".

Uma transação envolvendo vários arquivos é iniciada através do comando:

FS_OPEN[identificadores_dos_arquivos]

Essa operação retorna uma referência de conjunto e identificadores para todos os arquivos abertos. Esses identificadores mais a referência de conjunto são usados em posteriores operações em arquivos desse conjunto. Os arquivos do conjunto podem ser todos declarados de uma só vez, como parâmetro do comando **FS_OPEN**, contudo, existe a opção de adicionar arquivos ao conjunto usando-se o comando:

OPEN[referência_de_conjunto,nome_do_arquivo]

Para fechar (**Commit**) a transação é usado o comando:

FS_CLOSE[Parâmetros]

Passa-se como parâmetros deste comando a referência de conjunto (ident_trans) e a opção para o fechamento (completar ou abortar). Opcionalmente o comando

FS_COMMIT[referência_de_conjunto]

Pode ser usado para perpetuar todas as trocas feitas nos arquivos

daquela transação.

Para transação com apenas um arquivo os comandos:

OPEN e **CLOSE** ou **COMMIT**

delimitam uma transação atômica.

3) - **XDFS ("Xerox Distributed File System"** - [STURGIS 80])

O mecanismo de transações atômicas do **XDFS** é o mais geral dentre esses até aqui discutidos. Ele permite diferentes clientes atualizarem atomicamente vários arquivos em diferentes servidores. Essa generalidade é bastante desejada para aplicações em Banco de Dados.

No **XDFS** o início de uma transação é determinada explicitamente por uma operação:

LOGIN[endereço_do_servidor,ident_usuário,"password"]

Essa operação retorna um identificador de transação (**IDENT_TRANS**). Posteriores operações em arquivos são feitas usando-se o parâmetro **IDENT_TRANS**.

Uma transação atômica é concluída (COMMITED) através da operação:

CLOSE_TRANSACTION[ident_trans]

Essa operação devolve o resultado da tentativa de fechar a transação, que pode ser abortada ou concluida (COMMITED).

O comando:

ABORT[ident_trans]

aborta a transação em andamento identificada por ident_trans.

Para transações que envolvem mais que um servidor de arquivo o comando:

ADD_SERVER[ident_trans,endereço_do_servidor,ident_usuário,
"password"]

é usado para cada novo SA envolvido na transação. Esse comando devolve um ident_trans, que deve ser usado em todas as operações nesse servidor.

No comando de fechamento de transação (commit) com múltiplos servidores, o ident_trans passado como parâmetro tem que ser o devolvido pelo comando **LOGIN**, pois esse ident_trans representa o servidor coordenador da transação.

No **XDFS** existe uma comando que permite aos clientes consultarem o estado de suas transações atômicas:

FINAL_STATE[ident_trans]

que devolve um parâmetro informando se a referida transação está ativa, fechada (commited), abortada, ou se ela é desconhecida pelo servidor de arquivo.

2.2.2.3 - **Implementação**

Nessa seção serão discutidas as características de implementação de transações atômicas no que se refere aos dois aspectos da atomicidade da transação:

1) Atômica com respeito ao **acesso concorrente**: Para que Transações Atômicas sejam indivisíveis com respeito ao acesso concorrente, algum mecanismo tem que assegurar que a execução paralela de um conjunto de transações produza o mesmo resultado que seria obtido se as transações fossem executadas sequencialmente. O mecanismo básico, usado em Servidores de Arquivo, para resolver esse problema é o uso de **TRANCAS** (LOCKS - vide seção 3.3) que são aplicadas para reservar arquivos (ou subconjunto de arquivos) para uso de uma determinada transação.

Com o mecanismo de "trancas" disponível, usa-se geralmente um protocolo, chamado protocolo de trancamento de duas fases (**two-fase locking protocol**). Nesse protocolo a primeira fase consiste na transação adquirir todas as trancas necessárias e a segunda fase, consiste na liberação das trancas. Assim, na implementação de T.A. o SA mantêm todas as trancas necessárias à transação, somente liberando-as quando o cliente pede para fechar a transação, ou quando ela é abortada.

O Servidor de Arquivo gera trancas à medida que os clientes têm acesso aos dados. Um problema que pode advir com o uso de TRANCAS (locks) é a ocorrência de "**deadlocks**". Os Servidores de Arquivo em geral se utilizam de "time-out" associado as trancas

para resolver o problema de detecção de "deadlocks", resultando em transações abortadas pelo Servidor de Arquivo.

No SA **FELIX** [FRIDRICH 81] todos os arquivos usados numa transação podem ser declarados no início da transação, permitindo que o SA possa evitar a ocorrência de "deadlocks".

2) Atômica com respeito à **falhas**: Durante a execução de uma transação Atômica todas as atualizações nos arquivos tem que ser feitas de maneira que, se a transação for abortada pelo cliente ou pelo Servidor devido à falhas, o estado dos arquivos tem que permanecer o mesmo do início da transação. Por outro lado, depois que o cliente solicitou que a transação fosse concluída (commit) e o SA respondeu confirmando o fechamento da transação, as atualizações feitas pelo cliente têm que ser perpetuadas independentemente de "quedas" do Servidor.

Para implementação de Transações Atômicas com respeito à falhas, a solução geralmente adotada em Servidores de Arquivo é nunca alterar diretamente o bloco do disco que contém os dados. Sempre que operações de escrita são feitas pelo clientes, novos blocos (**shadow blocks**) são alocados, e as informações necessárias (**lista de intenções**) à fixação desses novos blocos na atual estrutura de armazenamento são gravadas no disco de uma maneira segura (vide redundância de disco - seção 3.4). Após o cliente ter pedido para fechar (**commit**) a transação, o Servidor de Arquivo usa um algoritmo que fará as atualizações. Esse algoritmo em geral, primeiro grava a lista de intenções no disco, em seguida marca um "flag" ("commit flag") que definirá o estado da transa-

ção. Uma vez esse "flag" esteja marcado, a transação estará decidida (**commited**), o fato é comunicado ao cliente, e oportunamente aquelas atualizações serão feitas, mesmo na ocorrência de "quedas" do Servidor de Arquivo.

No **CFS**, por exemplo, uma Transação Atômica está associada a um objeto e para cada objeto existe um "bit" de fechamento (commit). Dentro da tabela de alocação, onde é gravada a lista de intenções, para cada bloco de armazenamento existem quatro estados associados: Alocado, não alocado, intencionando alocar e intencionando liberar. Para cada bloco que é atualizado pelo SA, em decorrência de pedidos de escrita do cliente, o **CFS** executa os seguintes passos:

1) Escolhe um novo bloco não alocado e marca intencionando alocar;
2) Troca o estado do bloco "endereçado" para intencionando liberar;
3) "Escreve" no novo bloco.

Quando o cliente pede para fechar (**commit**) a transação, o **CFS** depois de gravadas as informações necessárias ao fechamento da transação, marca o "bit" de fechamento (**commit**). Com o "bit" de "commit" marcado, o **CFS** pode então remover as intenções:

1) Troca os estados de todos os blocos de intencionando alocar para alocado;
2) Troca os estados de intencionando liberar para não alo-

cado;

3) Desmarca o "bit" de "commit".

Se a transação for abortada por qualquer motivo, obviamente todos os blocos voltam ao seus estados anteriores (vide 3.5), ou seja, a lista de intenções é desfeita.

Em sistemas com múltiplos Servidores participando de uma T.A., o algoritmo de fechar uma T.A. é mais complexo, desde que, a ação de fechar tem que ser coordenada para evitar que uns fechem enquanto outros abortam.

2.2.3 - Mecanismos para Controle de Concorrência

Nos Servidores de Arquivos atualmente implementados, o mecanismo básico disponível para controle de concorrência a arquivos, é o uso de trancas ("locks"). Quase todos os Servidores de Arquivos, mesmo os que não implementam T.A. (ex: WFS [SWINEHART 79]), dispõem desse mecanismo.

Uma operação de "**tranca**" é executada, pelo cliente, com a finalidade de evitar que outros clientes acessem indevidamente certos arquivos (ou trechos de arquivo).

Em sua forma convencional, o mecanismo de tranca é "disparado" como efeito colateral de uma operação de abertura (OPEN) de arquivo, e nesse caso, geralmente usa-se trancas do tipo "**vários leitores/único escritor**", ou seja, vários clientes podem concorrentemente ler um mesmo arquivo, porém, para escrita, o acesso a um arquivo tem que ser exclusivo. Uma tentativa de abrir um arquivo trancado resulta em erro, geralmente sinalizado pelo Servidor de Arquivo.

Para evitar que arquivos fiquem trancados (LOCKOUT) indefinidamente, geralmente usa-se um "time-out" associado à tranca. Ou seja, uma tranca pode ser "**quebrada**" se ele "segura" um arquivo por determinado período de tempo.

Quando o cliente fecha (CLOSE) um arquivo, então a tranca é liberada.

Alguns Servidores permitem uma grande variedade de tipos de trancas (ex: FELIX [FRIDRICH 81]), de modo que o compartilhamento de arquivos é bastante facilitado.

Em Servidores de Arquivo como o XDFS [MITCHELL 82], o mecanismo de tranca é ainda mais geral, permitindo que trancas estejam associadas a trechos de arquivos. Portanto, podendo vários clientes compartilharem trechos distintos de um mesmo arquivo, resultando em um ambiente de compartilhamento mais generalizado. Ambiente esse, apropriado para certas aplicações, como utilização de Banco de Dados.

Citaremos, como exemplo, o mecanismo de tranca (locking) do "FELIX FILE SERVER". Existem no FELIX seis diferentes modos de acesso. Uma tranca está associada a uma arquivo e seu tipo (modo de acesso) é especificado quando o arquivo é aberto:

OPEN [ident_arq,...,**modo_de_acesso**]

Os modos de acesso (tipos de tranca) são:

READ COPY - É dado ao cliente uma cópia da mais recente versão do arquivo. O cliente somente pode "ler" a cópia. Outros clientes podem acessar o arquivo original livremente;

WRITE COPY - É dado ao cliente uma cópia da mais recente versão do arquivo. Alterações feitas na cópia não são passadas ao arquivo original. Outros clientes podem acessar o arquivo para leitura/escrita;

READ ORIGINAL - O cliente pode acessar a última versão do arquivo, apenas para leitura. Não é permitido que outro cliente concorrentemente escreva no arquivo;

WRITE ORIGINAL - O cliente escreve na última versão do arquivo. Somente clientes com "WRITE COPY" e "READ COPY" podem acessar o arquivo concorrentemente;

READ EXCLUSIVE - O cliente tem acesso exclusivo ao arquivo, podendo apenas fazer operações de leitura;

WRITE EXCLUSIVE - O cliente tem acesso exclusivo ao arquivo, podendo fazer operações de leitura e escrita.

Toda vez que um cliente pede para abrir um arquivo, se existe conflito de tranca (lock), o seu pedido de abertura é colocado numa fila. O diagrama a seguir esclarece o comportamento desse mecanismo. Um cliente só tem acesso a um arquivo se não existe conflito com outros já acessando, ou se não tem conflitos com pedidos anteriormente na fila.

Modos de Acesso de clientes atuais ou enfileirados

Pedidos de OPEN	LIVRE	RC	WC	RO	WO	RX	WX
RC	acessa	acessa	acessa	acessa	acessa	enfila	enfila
WC	acessa	acessa	acessa	acessa	acessa	enfila	enfila
RO	acessa	acessa	acessa	acessa	enfila	enfila	enfila
WO	acessa	acessa	acessa	enfila	enfila	enfila	enfila
RX	acessa	enfila	enfila	enfila	enfila	enfila	enfila
WX	acessa	enfila	enfila	enfila	enfila	enfila	enfila

Legenda: LIVRE - Nenhum cliente está tentando acessar o arquivo;
RC - READ COPY;
WC - WRITE COPY;
RO - READ ORIGINAL;
WO - WRITE ORIGINAL;
RX - READ EXCLUSIVE;
WX - WRITE EXCLUSIVE;
ACESSA - O cliente acessa o arquivo;
ENFILA - O pedido do cliente é colocado na fila.

Diagrama extraido de [FRIDRICH 81].

2.2.4 - Redundância de Disco

Quando arquivos são alterados por clientes dentro de uma transação atômica, os dados são gravados de modo que seja possível reverter o estado dos arquivos para o do início da transação. Se acontesse uma falha de hardware, por exemplo, causada pela queda de energia durante uma operação de escrita, resultando em erro de gravação, temos dois casos a considerar:

1) Se o bloco do disco afetado corresponde a dados do cliente : A transação pode ser abortada e o conteúdo dos arquivos volta a ser o de antes da transação;

2) Se o bloco afetado pertence a lista de intenções ou à estrurura de dados que define a organização interna dos arquivos, ou à estrutura de dados para gerenciamento de armazenagem (ex: mapa do disco): O dano causado pelo erro de gravação compromete de maneira irreversível a integridade dos dados no Servidor de Arquivos.

Para resolver esse problema, cria-se o que se chama de "redundância de disco". Ou seja, as informações seriam gravadas no disco de maneira redundante, de modo que o bloco corrompido possa ser reconstruído a partir de informações redundantes.

Essa redundância é mais geral à medida que, aumentando-se a área afetada, consegue-se a recuperação da mesma.

A seguir são dados dois exemplos de Servidores atualmente

implementados, O CFS e o XDFS:

XDFS - O XDFS se utiliza do conceito de memória estável para implementar redundância de disco. Memória estável é implementada por "software", usando-se as propriedades já disponíveis em controladores de discos convencionais. Memória estável garante que, se durante uma operação de escrita de um bloco do disco, houver falha, ou o bloco é gravado com sucesso ou ele permanece intacto.

Cada bloco lógico do disco é representado por dois blocos físicos na memória estável. Assim, cada bloco lógico que é escrito, resulta em dois blocos físicos sendo alterados. A segunda cópia do bloco somente é alterada depois da primeira ter sido alterada com sucesso.

CFS - No CFS, além de mapas de objetos, existe uma estrutura chamada de mapa do cilindro. Um bloco por cilindro é reservado para conter informações sobre a estrutura de objetos naquele cilindro. Um mapa de cilindro é um "array" indexado por número de setor. Cada entrada representa um bloco do cilindro e contém o estado de alocação (alocado, não alocado, intencionando alocar e intencionando liberar), o identificador do objeto ao qual ele faz parte e a posição do bloco na árvore do objeto. A entrada do mapa do cilindro correspondente ao bloco raiz de uma árvore, contém também o bit de "commit" do objeto.

Com essa estrutura redundante, cada bloco é reconstruível. Se um bloco do mapa do objeto (vide 3.8) é destruído, o mapa do cilindro é usado para refazê-lo. Do mesmo modo, se o mapa do ci-

lindro é destruído, ele é recuperado percorrendo-se as árvores dos objetos.

A semelhança de armazenamento estável, o mapa do cilindro somente é alterado depois da árvore do objeto ter sido alterada sem ocorrência de erro.

2.2.5 - Recuperação após "quedas" do Servidor de Arquivo

Quando o Sevidor "cai" por falha de hardware, ou mesmo por "furos" de software, provavelmente algumas Transações de Clientes estarão em andamento e outras já concluídas (commited), embora, não necessariamente com as alterações nos arquivos efetivadas. Se quando da "queda", o cliente não tinha ainda pedido para fechar a transação, ou se, o cliente tinha pedido, mas o SA não tinha decidido (commited) a transação, então a transação será **abortada** e a **lista de intenções** desfeita. Se o SA já tinha cinfirmado para o cliente o fechamento (commitment), entretanto, não tinha executado as alterações, então, no processo de **recuperação** o SA tem que perpetuar aquelas alterações, executando a lista de intenções.

No processo de recuperação, antes das listas de intenções serem concluídas ou desfeitas, o SA verifica se a "queda" causou o corrompimento de algum bloco do disco. Se houve bloco com erro de gravação e esse bloco pertence a área de armazenagem redundante, então as estruturas redundantes (vide 3.4 - redundância de disco) são acionadas no sentido de recuperar este bloco.

No **CFS**, por exemplo, o processo de recuperação consiste primeiro em verificar se houve erro de gravação no mapa do cilindro, ou num bloco de mapas de objetos, em caso afirmativo é feita a recomposição desse bloco. O passo seguinte, consiste em percorrer as tabelas de alocação de bloco que contêm a lista de intenções a fim de determinar o estado de alocação de cada bloco. Se existem blocos em estado de **intencionando alocar** ou **intencionando liberar**, o "bit" de fechamento (commit) associado ao objeto, ao

qual pertence o bloco, irá determinar o estado final do bloco (alocado ou não alocado). A figura 3 representa o diagrama de transição de estados.

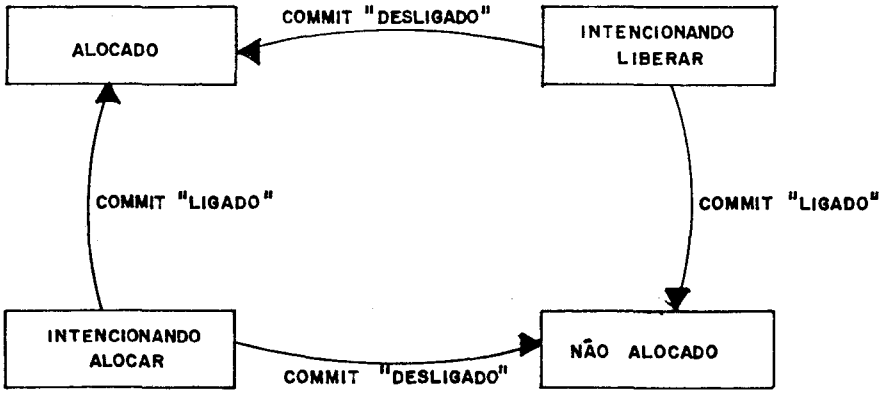

Figura 3 - Diagrama de transiçao de estados dos blocos [DION 80], no processo de recuperação do CFS.

Em alguns Servidores, o processo de recuperação é feito imediatamente após a reinicialização, em outros, somente quando necessário, devido ao acesso a dados afetados pela "queda".

2.2.6 - Controle de Acesso

As duas principais técnicas usadas no controle de acesso a arquivos, são "**lista de controle de acesso**" e "**capability**" [SALTZER 75]. Num esquema de "lista de controle de acesso", para cada arquivo existe associada uma lista de clientes que podem usá-los. Em sistemas que fazem controle de acesso baseado em "capability", o acesso a um arquivo depende unicamente do cliente apresentar uma "capability" válida para aquele arquivo, não necessitando o cliente ser identificado junto ao Servidor.

Uma vantagem intrínseca do uso de "capabilities" em Servidores de Arquivos, é que vários clientes podem naturalmente participar de uma única transação atômica, bastando somente, que os clientes cooperem entre si mandando a "capability" necessária.

Em Servidores de Arquivos que usam "lista de controle de acesso", é necessário um mecanismo especial para permitir que vários clientes paticipem de uma mesma Transação Atômica. Um outro problema relacionado com "lista de controle de acesso", é determinar a identidade de um dado cliente, desde que, podemos ter muitos clientes, vários num mesmo computador.

Para Servidores de Arquivo (ex: "**Woodstock File Server**") sem nenhum mecanismo de controle de acesso disponível (possibilitando a qualquer máquina na rede, que possa se comunicar com o SA, ter total acesso a todas as operações em todos os arquivos), **Needham** [NEEDHAM 79] propõe um método para inclusão de controle de acesso baseado em "capability".

2.2.7 - Modelo de Arquivo

Consideraremos nessa seção, o modelo de arquivo no que se refere ao acesso aos dados e a estrutura do arquivo "vista" pelo cliente.

Os clientes de Servidores de Arquivos em geral "enxergam" arquivos como um conjunto de blocos de dados de tamanho fixo. Esses blocos são geralmente estruturados como uma sequência, cujos conteúdos são "lidos" e "gravados" a partir de acessos aleatórios. Em alguns Servidores (ex: CFS), clientes podem "ler" ou "gravar" qualquer subconjunto do arquivo numa única operação.

A seguir são mostradas as operações de acesso a arquivos correspondente a três Servidores atualmente implementados:

XDFS: No XDFS um arquivo é organizado como uma sequência aleatória de bytes. O cliente pode acessar um subconjunto de bytes dentro de um arquivo.

Operações:

Leitura: READ[indent_arq.,deslocamento,número_de_bytes]

Essa operação devolve o número de bytes especificado na posisão indicada por "deslocamento".

Escrita: WRITE[ident_arq.,deslocamento,número_bytes,dados].

CFS : No CFS um arquivo é organizado como uma sequência de registros (cada um com 16 "bits") de acesso aleatório. Podendo o cliente em uma única operação transferir um conjunto contínuo de registros.

Operações:

Leitura: READ[ident_arq.,deslocamento,tamanho]

Essa operação devolve uma quantidade de registros contínuos, especificada por "tamanho", sendo a primeiro registro da sequência localizada pelo parâmetro "deslocamento".

Escrita: WRITE[ident_arq.,deslocamento,tamanho,dados].

WFS: Os arquivos são "vistos" pelos clientes como uma sequência de páginas de tamanho fixo. O Cliente acessa uma determinada página (bloco).

Operações:

Leitura: READ_PAGE[ident_arq.,número_página].

Escrita: WRITE_PAGE[ident_arq.,número_página,dados].

2.2.8 - Estrutura de Armazenamento

Para o acesso a arquivos, Servidores de Arquivo em geral se utilizam de um identificador numérico (ident_arq), que muitas vezes é constituído a partir do endereço inicial do disco onde ele reside. Usuários de sistemas de arquivos convencionais costumam identificar seus arquivos por meio de nomes alfanuméricos (exs: teste, alfa, dat03, etc.). Assim, clientes de Servidores de Arquivo, geralmente têm que implementar um serviço de diretório para "mapear" nomes de arquivos em identificadores (ident_arq) usados internamente pelo SA para acessar arquivos. Em alguns Servidores (ex: "FELIX FILE SERVER" [FRIDRICH 81]) esse serviço já é oferecido na interface do cliente.

Nos Servidores de Arquivo, geralmente os arquivos são estruturados como **árvores**, com alguns blocos contendo "ponteiros" para outros blocos e nas folhas da árvore blocos de dados dos arquivos. Além dos dados, algumas informações sobre o arquivo também são armazenadas (data de criação, tamanho do arquivo, etc.). Essas informações geralmente são armazenadas num cabeçalho do arquivo que pode se localizar, por exemplo, num bloco de "ponteiros".

Veremos a seguir a estrutura de armazenamento de dois Servidores atualmente implementados:

O CFS [DION 80] armazena dois tipos de objetos (arquivos e índices), cada objeto é identificado por um identificador único (IDU). Um IDU é uma "capability" [SALTZER 75] para o objeto, e é constituida de 64 bits, sendo 32 bits um número aleatório, e os

outros 32 bits o endereço do disco onde reside o objeto.

Um objeto é criado a partir de uma operação explicita do cliente ("create_index" ou "create_file").

Um arquivo é uma sequência de registros, 16 bits cada, acessados aleatoriamente. Um índice é uma lista de IDU's, podendo conter qualquer IDU.

Os clientes do CFS "enxergam" o armazenamento como um GRAFO DIRECIONADO, cujos vértices são arquivos e índices. Existe um **índice raiz** cuja permanência é garantida pelo CFS. Os outros objetos somente existem enquanto existir um **caminho** do vértice raiz até eles. A seguir é mostrado uma possível organização do armazenamento.

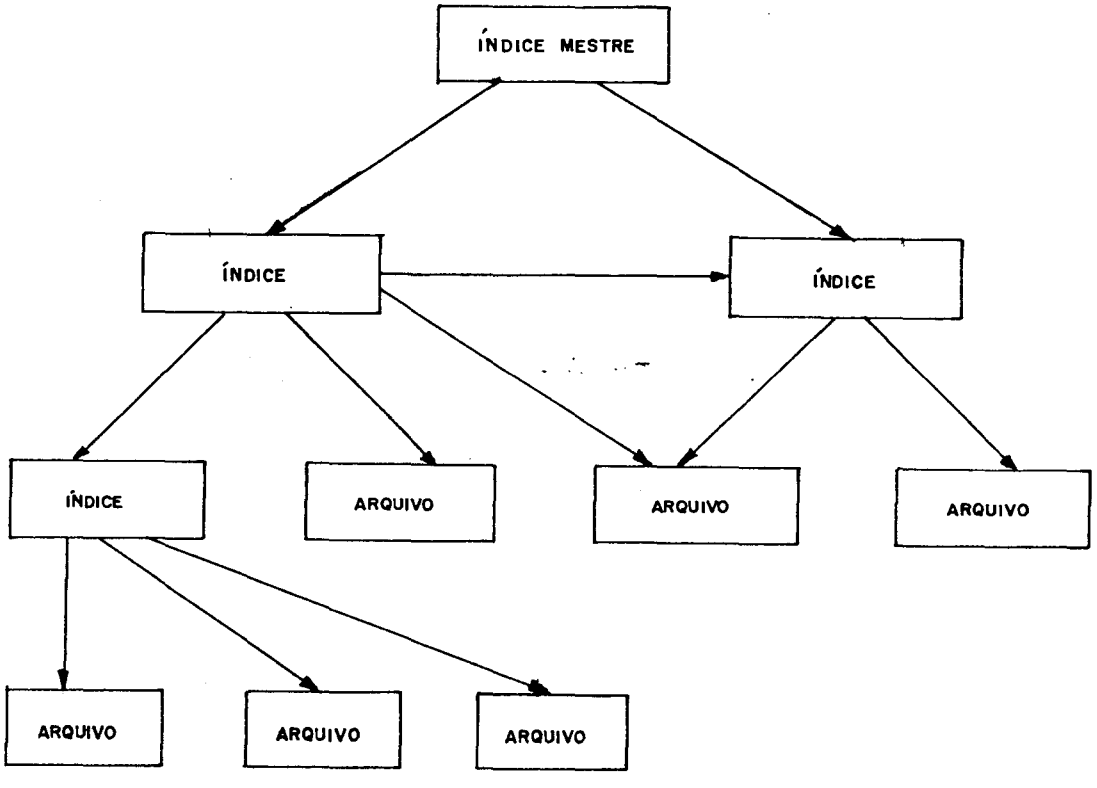

Figura 4 - Uma possível organização para o CFS

Quando um objeto deixa de existir (não mais existe caminho do vértice raiz até ele), o Servidor detecta através de um contador de referências relacionado com o objeto. Cada aresta incidente no objeto, acresce seu contador de referências de um. Assim, se o contador de referências chega a zero o objeto já não mais existe. Isso é verdadeiro, se o GRAFO for também uma ARVORE, ou seja, não contêm CICLOS. Se são permitidos ciclos, somente a técnica de contador de referência não é suficiente. Então, é usado um "coletor de lixo" [GARNETT 80] para periodicamente liberar armazenagem correspondente à objetos não acessíveis.

Cada objeto no CFS é estruturada como uma árvore de blocos de disco, que pode ter até três níveis, dependendo do tamanho do objeto. Assim quando um objeto (arquivo ou índice) é criado, ele contém apenas um bloco de disco vazio, "apontado" pelo seu IDU.

Os blocos de disco são de dois tipos:

Blocos de Mapa: Definem a estrutura do objeto (contém apenas endereços de disco);

Blocos de Dados: mantém o conteúdo do objeto.

A seguir é dado um exemplo [DION 80] da estrutura de um objeto de dois níveis:

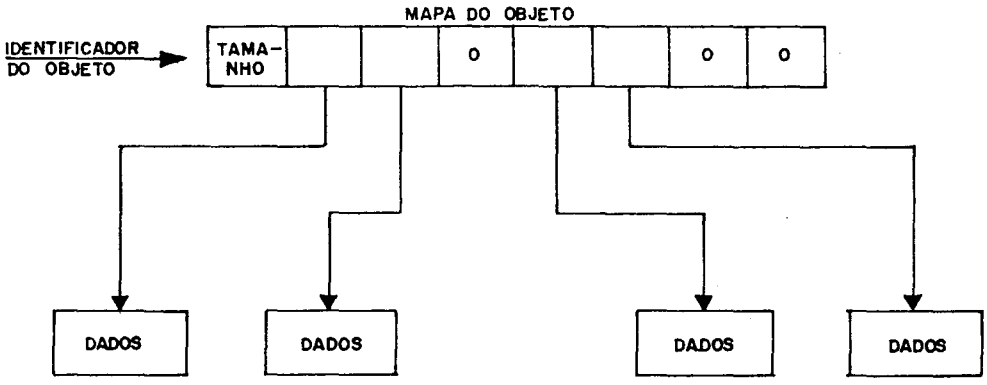

Figura 5 - Estrutura de um objeto de dois níveis no CFS.

No **WFS** [SWINEHART 79] um arquivo é identificado por um número inteiro de 32 bits (IDA). Não existe uma estrutura convencional de diretório. Um cliente do WFS pode implementar um sistema de arquivo e usar um IDA como identificador de, por exemplo, um "disco virtual".

Cada arquivo é um conjunto de blocos (páginas) de disco que podem ser endereçadas randomicamente.

O WFS usa uma tabela "**hash**", que é implementada como um arquivo de tamanho fixo de endereço conhecido, e usada para "mapear" IDA's em seus respectivos mapas de blocos (páginas). O mapa de bloco é usado para traduzir um número de bloco do cliente num endereço físico do disco. Um mapa de bloco pode ter até dois níveis. A figura a seguir ilustra a estrutura de armazenamento do WFS.

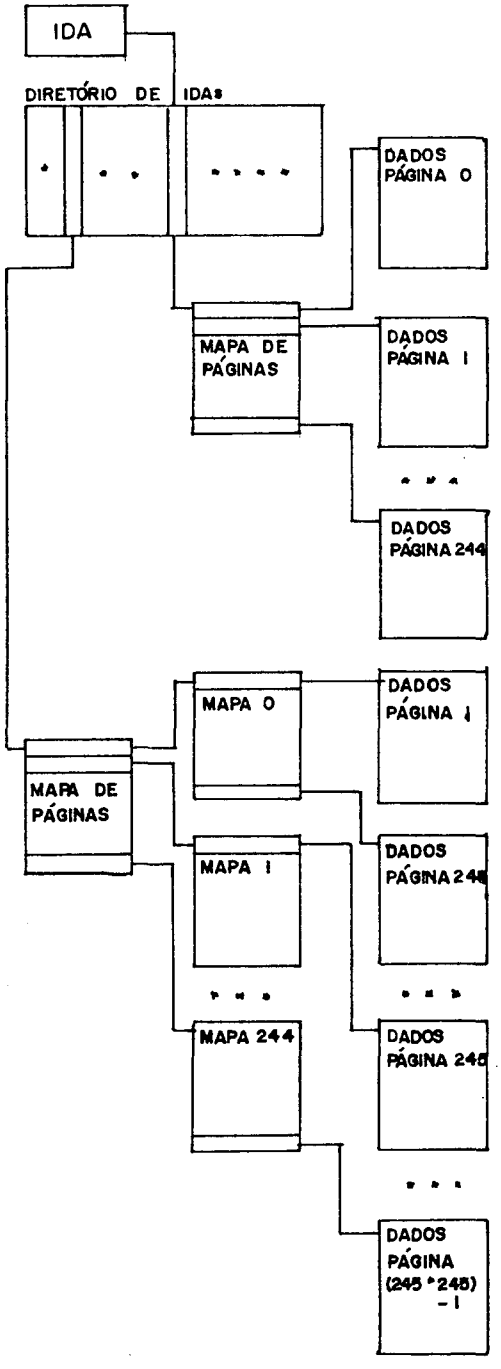

Figura 6 - Estrutura de armazenamento do WFS.

3.0 SISTEMA PROPOSTO

3.1 Introdução

Nesta tese é proposto um SA visando clientes de "baixo custo". A denominação "baixo custo", por um lado, deriva do fato de visarmos clientes "rodando" em máquinas baratas, tipicamente micro-computadores de 8 bits com uma unidade de disquete. Por outro lado, essa denominação também é usada em oposição à clientes que usariam o SA como um disco remoto, sendo forçados a implementar suas próprias estruturas, o que os tornariam clientes "caros".

A seguir são descritas características relativas ao Servidor proposto e à implementação do protótipo.

Servidor voltado para um específico sistema de arquivo:

O fato de se propor um SA voltado para um específico sistema de arquivo é justificado principalmente pelo fato de estarmos visando um ambiente computacional onde as ET's seriam micro-computadores de 8 bits usando o mesmo sistema operacional. Assim é desejável que tenhamos clientes gastando poucos recursos computacionais, ou seja, clientes que não necessitem implementar estruturas de alto nível encontradas em sistemas de arquivos convencionais. Estes clientes são os sistemas operacionais das ET's, que usando um SA que dispõe de todas as funções de alto nível de um sistema de arquivo convencional, gastam "código" praticamente para comunicação com o SA e pequenas compatibilizações entre o sistema operacional residente (CP/M) e as mensagens trocadas com

o Servidor de Arquivo.

Compatibilidade com CP/M:

O sistema operacional CP/M tornou-se um padrão para micro-computadores de 8 bits, já existindo uma quantidade muito grande de programas aplicativos disponíveis. Em vista disso, Um outro objetivo que se almeja, é compatibilidade com CP/M, de modo que programas anteriormente escritos sobre CP/M "rodariam" no sistema operacional da Estação de Trabalho usando o Servidor de Arquivo, sem necessidade de adaptações (vide seção 3.6.2). Essa compatibilidade é as vezes difícil, já que CP/M não prevê nenhum controle para acesso concorrente a arquivos.

Transações Atômicas:

Considerando-se o fato de que em algumas aplicações, como o reprocessamento de uma compilação, a reprodução de resultados seria trivial e principalmente considerando-se que o Hardware do SA (micro de 16 bits e memória em disco, tipicamente com 10 mega-bytes) não é suficientemente "poderoso" para suportar o "over-head" causado pelo mecanismo de Transações Atômicas, sem contudo prejudicar o desempenho do sistema, optou-se por não implementar Transações Atômicas, ficando a cargo do programa de aplicação a implementação de mecanismos que possibilitem atualizações atômicas.

É importante notar que o protocolo SA/ET foi implementado de

tal forma, que quando se recebe um sinal de **operação efetuada** do Servidor de Arquivo, tem-se a certeza de que a integridade do armazenamento é garantida, pois as alterações no disco sempre precedem o referido sinal.

Sincronização de Acesso Concorrente:

Para que usuários em Estações de Trabalho distintas, pudessem ter acesso a arquivos concorrentemente, considerou-se indispensável algum mecanismo para sincronização do acesso concorrente. O mecanismo usado é a TRANCA (LOCK). Foram implementados no protótipo, cinco diferentes tipos de **modo de acesso (tipos de tranca)** - Vide operação de abertura de arquivo. Para programas anteriormente escritos em CP/M, o acesso concorrente é feito de modo que, o primeiro usuário que acessa (abre) o arquivo tem acesso exclusivo para escrita, podendo concorrentemente outros usuários "lerem" o arquivo. Qualquer tentativa de abertura de arquivo que cause conflitos, é sinalizada com um erro.

Estrutura de Armazenamento no Servidor de Arquivo:

Apesar da semelhança entre a estrutura de armazenamento do SA e o CP/M, foi feito algum esforço no sentido de não herdar algumas ineficiências encontradas no CP/M, decorrentes de sua estrutura interna. Uma primeira providência foi a de dispor o diretório totalmente na memória. Entradas de diretório de um mesmo arquivo são encadeadas formando uma lista ligada. Também todas as

entradas do diretório que estão livres formam uma lista de entradas disponíveis.

Foram criados registros de ativação para arquivos abertos. Esses registros são identificados por números inteiros, que são fornecidos ao usuário quando da abertura de um arquivo. Esse número funciona como uma "capability", o usuário tem que fornecê-lo para futuros acessos ao arquivo.

Explorando características como: "Localidade de referência" a dados e "Sequencialidade de acessos", foi implementado um **Cache** visando otimizar o acesso ao disco. Esse cache é organizado como um "conjunto-associativo", usa política para substituição LRU e política para atualizações do disco em acessos de escrita tipo "write-through".

Protótipo:

O objetivo central deste trabalho é o desenvolvimento do SA. Para implementação do protótipo procurou-se utilizar uma rede local de baixo custo e fácil implementação. Para isso utilizamos portas seriais disponíveis em um compatível com o IBM-PC (PC-XT da ITAUTEC), fazendo ligações ponto-a-ponto entre o SA (IBM-PC) e as Estações de Trabalho (Micros CP/M de 8 bits - I7000/ITAUTEC). Formando o que se chama de rede local estrela. É importante notar que os protocolos implementados são de propósito específico para essa aplicação.

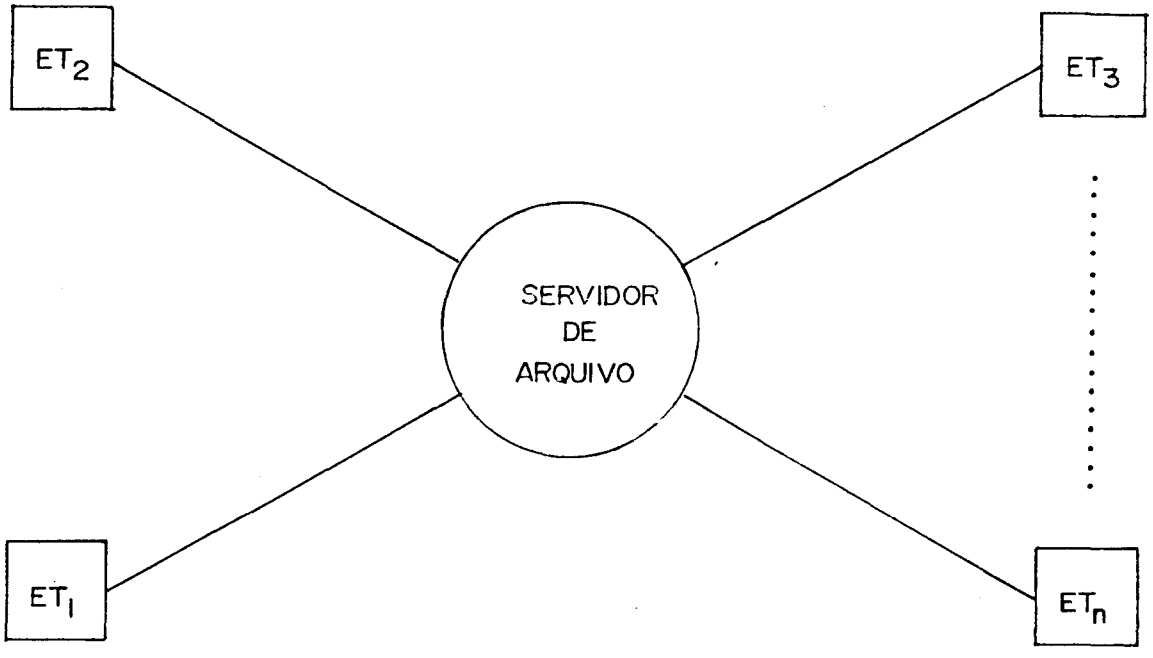

Figura 7 - Rede Local Estrela

O protótipo desenvolvido consiste de:

- **Servidor de Arquivo**

  . Serviço de gerenciamento de arquivos (criação, leitura, escrita, proteção, etc);

  . Protocolo de comunicação com as Estações de Trabalho.

- **Sistema operacional das ET's adaptado**

O sistema opereracional das ET's (compatível com CP/M) foi alterado para permitir acesso ao SA. Para isso foi incluído

um novo módulo no sistema operacional (NOVO-BDOS). Também foram feitas outras modificações visando compatibilizações com o sistema operacional usado (SIM/M), em virtude da não disponibilidade do código fonte de tal sistema operacional.

Operação:

Para que o usuário, na ET, possa acessar o SA, basta que ele selecione uma das unidades lógicas que representam o SA, para ser o "drive" corrente. No protótipo apenas a unidade "A" está associada ao SA. A operação de selecionar o drive "A" verifica se o SA está no "ar" e em caso negativo, uma mensagem notificará a ausência do Servidor de Arquivo.

O SA atende à pedidos das ET's usando uma política circular. Uma operação do SA é executada por vez, para cada Estação de Trabalho.

3.2 - Interface do Cliente

A Interface do Cliente é formada pelo conjunto de operações através da qual se tem acesso ao Servidor de Arquivo.

As operações serão descritas em termos de mensagens, onde **ENTRADA** descreve a mensagem enviada ao Servidor, correspondente a uma operação e **SAÍDA** descreve a resposta do Servidor que corresponde ao resultado da operação.

1) **SELECIONA UNIDADE**

Esta operação carrega o diretório na memória e monta o mapa de alocação do disco da unidade selecionada.

Entrada: 1º byte : Código da operação (00);
2º byte : Código da unidade. 0 - Unidade "A";
1 - Unidade "B";
2 - Unidade "C";
3 - Unidade "D".

Saída: 1º byte : Operação efetuada (03) ou
Erro de acesso ao disco (255).

OBS: Cada unidade lógica corresponde a um disco (winchester ou floppy) ligado ao SA. No protótipo está implementado apenas uma unidade, a unidade "A", que corresponde a um "floppy disk".

2) ABRE ARQUIVO

Esta operação abre um arquivo para um determinado modo de acesso (tipo de TRANCA), criando um registro de ativação no SA para aquele arquivo aberto.

Entrada: 1º byte : Código da operação : (01);

2º byte : Modo de Acesso (tipo de TRANCA) :

0 - **Acesso padrão** - O primeiro usuário a abrir o arquivo pode "escrever", os demais podem paralelamente apenas "ler";

1 - **Leitura somente** - Acessa somente para "leitura", sem trancas;

2 - **Leitura-escrita** - Acessa para "escrita", podendo paralelamente ser "lido" por outros usuários;

3 - **Multi-escrita** - Compartilhamento sem restrições;

4 - **Leitura-exclusiva** - Somente leitura com tranca para escrita;

5 - **Escrita-exclusiva** - acesso exclusivo.

3º byte : Unidade lógica;

4º ao 11º byte : Nome do Arquivo;

12º ao 14º byte : Tipo do arquivo;

15º byte : Número de extensão (entrada do diretório);

16º byte : Registro atual (deslocamento dentro do último registro dentro do último bloco de alocação alocado).

Saída: 1º byte : Operação efetuada (03) ou
Modo de acesso conflitante (251) ou
Arquivo protegido contra escrita (252) ou
Não existe registro de ativação disponível (253) ou
Arquivo não encontrado (254) ou
Erro de acesso ao disco (255).

2º byte : Identificador de Registro de Ativação para o arquivo aberto (0 a 255).

3) FECHA ARQUIVO

Esta operação fecha um arquivo anteriormente aberto, atualizando o diretório caso o arquivo tenha sido alterado e liberando o registro de ativação para aquele arquivo aberto.

No Servidor de Arquivo, todos os arquivos abertos têm que ser fechados. Alternativamente, a operação ENCERRA_SESSAO pode ser usada pelo usuário para liberar registros de ativação de arquivos abertos.

Entrada: 1º byte : Código de operação (02);
2º byte : Identificador do Registro de Ativação (número inteiro de 0 a 255);

Saída: 1º byte : Operação efetuada (03) ou
Arquivo não aberto (254) ou
Registro de ativação inexistente (253) ou
Erro de acesso ao disco (255).

4) **PROCURA BCA** (Bloco de Controle de Arquivo)

Esta operação tenta encontrar a primeira ocorrência do nome de determinado arquivo. O arquivo é identificado pelo seu nome, extensão e número de entrada (extent). Se o campo "drive-lógico" contiver o caractere "?", será procurado o arquivo em todas as entradas do diretório, pertencentes a qualquer usuário, alocada ou não.

ENTRADA: 1º byte : Código da operação (03);
2º byte : Unidade lógica (1="A", 2="B", etc.);
3º ao 10º byte : Nome do arquivo;
11º ao 13º byte : tipo do arquivo (extensão);
14º byte : Número de extensão (entrada do diretório)

SAÍDA: 1º byte : Arquivo encontrado (03) ou
arquivo não encontrado (255);
2º ao 31º byte: Entrada de Diretório Procurada.
32º byte : Ponto de partida para uma próxima busca desse arquivo no diretório.

5) PROCURA PRÓXIMO BCA

Esta operação tenta encontrar a próxima ocorrência do nome de um determinado arquivo no diretório, a partir da última ocorrência verificada, através da operação 4, ou outra operação 5. O caractere "?", quando usado no campo "drive-corrente" irá fazer com que, a busca seja feita em todas as entradas do diretório, pertencentes a qualquer usuário, alocadas ou não.

ENTRADA: 1º byte : Código da operação (04);
2º byte : Unidade lógica ("1"="A", 2="B", etc.);
3º ao 10º byte : Nome do arquivo;
11º ao 13º byte : Tipo do arquivo (extensão);
14º byte : Número de extensão (entrada de diretório);
15º byte : Ponto de início para busca no diretório.

SAÍDA: 1º byte : Arquivo encontrado (03) ou
Arquivo não encontrado (255);
2º ao 31º byte : Entrada de diretório procurada;
32º byte : Ponto de partida para próxima busca desse arquivo no diretório.

6) ELIMINA ARQUIVO

Esta operação exclui todas as ocorrências de um determinado arquivo presentes no diretório. Arquivos do tipo "read-only" e arquivos abertos não são eliminados.

ENTRADA: 1º byte : Código da operação (05);

2º byte : Unidade lógica;

3º ao 10º byte : Nome do arquivo;

11º ao 13º byte : Extensão do nome do arquivo (tipo).

SAÍDA: 1º byte : Arquivo eliminado (03) ou

Arquivo não encontrado (255) ou

Alguns arquivos não puderam ser apagados (254) ou

Arquivos não puderam ser apagados (253) ou Erro de acesso ao disco (252).

7) **LEITURA SEQUENCIAL**

Esta operação faz com que o próximo registro (128 bytes) sequencial de um arquivo, aberto por "CRIA ARQUIVO" ou "ABRE ARQUIVO", seja lido. Esta operação também incrementa o campo que indica qual o número do próximo registro acessado sequencialmente.

ENTRADA: 1º byte : Código da operação (06);

2º byte : Identificador do registro de ativação (número inteiro de 0 a 255).

SAÍDA: 1º byte : Registro lido (03) ou

Erro de acesso ao disco (253) ou

Registro de ativação inexistente (254) ou

Fim de Arquivo (255);

2º ao 129º byte : Setor de 128 bytes.

8) **ESCRITA SEQUENCIAL**

Esta operação faz com que seja gravado o próximo registro de um arquivo de acesso sequencial, aberto através das operações "ABRE ARQUIVO" ou "CRIA ARQUIVO". Também é incrementado o campo que indica o número do próximo registro a ser gravado.

ENTRADA: 1º byte : Código da operação(07);
2º byte : Identificador do registro de ativação;
3º ao 130º byte: Setor de 128 bytes.

SAÍDA: 1º byte : Registro gravado (03) ou
Disco Cheio (255) ou
Registro fora do arquivo (254) ou
Erro de gravação (252) ou
Arquivo de escrita proibida (251) ou
Registro de ativação inexistente (253).

9) **CRIA ARQUIVO**

Esta operação cria um novo arquivo no disco. O arquivo é inicializado para futuros acessos do mesmo modo que é feito quando de uma operação de abertura de arquivo. O SA não verifica duplicidades de nomes no diretório, ou seja, podem existir mais de um arquivo com nomes iguais. No SA, a operação CRIA ARQUIVO gera um registro de ativação para o novo arquivo. Assim, se paralela-

mente for executado uma operação ABRE ARQUIVO para o mesmo arquivo, um outro registro de ativação será gerado para o referido arquivo, sendo necessário, portanto, duas operações FECHA ARQUIVO, para que os dois registros de ativação sejam liberados e o arquivo finalmente fechado.

ENTRADA: 1º byte : Codigo da operação (08);

2º byte : Tipo de Acesso:

0 - **Acesso Padrão** - Acessa o arquivo com direito para escrita, outros usuários podem concorrentemente ler o arquivo;

1 - **Leitura-Somente** - Acessa somente para leitura, sem trancas;

2 - **Leitura-escrita** - Acessa para escrita, podendo paralelamente ser lido por outros usuários;

3 - **Multi-escrita** - Compartilhamento sem restrições;

4 - **Leitura-exclusiva** - Somente leitura com tranca para escrita;

5 - **Escrita-exclusiva** - Acesso exclusivo.

3º byte : Unidade lógica;

4º ao 11º byte : Nome do arquivo;

12º ao 14º byte : tipo do arquivo (extensão do nome).

Saída: 1º byte : Arquivo criado (03) ou
Diretório cheio (254) ou
Não há registros de ativação disponíveis (255) ou
Disco com problemas para gravação (253);
2º byte : Identificador do registro de ativação
(Número inteiro entre 0 e 255).

10) **TROCA NOME DE ARQUIVO**

Esta operação realiza a troca de um nome de arquivo presente no diretório.

ENTRADA: 1º byte : Código da operação (09);
2º byte : Unidade lógica;
3º ao 10º byte : Nome anterior do arquivo;
11º ao 13º byte : Tipo anterior do arquivo;
14º ao 21º byte : Nome novo do arquivo;
22º ao 24º byte : Tipo novo do arquivo.

SAÍDA: 1º byte : Operação realizada (03) ou
Arquivo não encontrado (254) ou
Disco com problemas para gravação (255).

11) **ALTERA NÚMERO DE USUÁRIO CORRENTE**

O armazenamento no SA está, para os clientes, dividido lógicamente em N grupos de usuários. No protótipo foram implemen-

tados 16 grupos distintos (numerados de 0 a 15). Esse comando altera o número de usuário corrente para uma dada estação de trabalho.

ENTRADA: 1º byte : Código da operação (10);
2º byte : Número de usuário.

SAíDA: 1º byte: Operação efetuada (03), ou
Número de usuário inválido (255).

12) **ENCERRA SESSÃO**

Esta operação faz com que todos os registros de ativação correspondentes a operações de abertura de arquivos, feitas a partir de determinada Estação de Trabalho, sejam liberadas. Esta operação corresponde ao fechamento de todos os arquivos abertos por uma determinada Estação de Trabalho.

ENTRADA: 1º byte : Código da operação (11).

SAíDA: 1º byte : Operação efetuada (03) or
Erro de acesso ao disco (255).

13) **LE REGISTRO RANDOMICAMENTE**

Esta operação realiza a leitura de um registro de 128 bytes, em um arquivo de acesso randômico.

ENTRADA: 1º byte : Código da operação (12);

2º byte : Identificador de registro de ativação;

3º byte : Byte menos significativo da chave randômica;

4º byte : Byte mais significativo da chave randômica;

SAÍDA: 1º byte : Operação efetuada (03) ou
Registro de ativação inexistente (255) ou
Busca após fim físico do disco (254) ou
Busca de uma extensão não escrita (253) ou
Leitura de dado não escrito (252) ou
Erro de acesso ao disco (251);

2º ao 119º byte : Registro lido.

14) **ESCREVE REGISTRO RANDOMICAMENTE**

Esta operação grava um registro de 128 bytes, em um arquivo, randômicamente.

ENTRADA: 1º byte : Código da operação (13);

2º byte : Identificador do registro de ativação do arquivo;

3º byte : Byte menos significativo da chave de acesso;

4º byte : Byte mais significativo da chave de acesso;

5º ao 132º byte : Registro a ser gravado.

SAÍDA: 1º byte : Operação efetuada (03) ou;
Registro de ativação inexistente (255) ou
Busca após final físico do disco (254) ou
Diretório cheio (não conseguiu criar nova extensão) (253) ou
Erro de acesso ao disco (252) ou
Disco cheio (251) ou
Arquivo de escrita proibida (250).

15) **CALCULA TAMANHO VIRTUAL DO ARQUIVO**

Esta operação calcula o tamanho do arquivo de acesso randômico, ou seja, devolve o valor correspondente ao número do último registro somado de 1.

ENTRADA: 1º byte : Código da operação (14);
2º byte : Unidade lógica;
3º ao 10º byte : Nome do arquivo;
11º ao 13º byte : tipo do arquivo.

SAÍDA: 1º byte : Operação efetuada (03) ou
Arquivo não encontrado (255);
2º byte : Byte de overflow (É preenchido com 01 se o tamanho for igual a 32.768);
3º byte : Byte mais significativo do tamanho;
4º byte : Byte menos significativo do tamanho.

16) **TORNA O ACESSO RANDÔMICO**

Esta operação devolve a chave randômica do último registro acessado sequencialmente.

ENTRADA: 1º byte : Código da operação (15);
2º byte : Identificador do registro de ativação do arquivo.

SAÍDA: 1º byte : Operação efetuada (03) ou
Registro de ativação inexistente (255);
2º byte : Byte menos significativo da chave randômica;
3º byte : Byte mais significativo da chave randômica.

17) **ESCREVE REGISTRO RANDÔMICAMENTE E PREENCHE COM ZEROS O BLOCO DE ALOCAÇÃO**

Esta operação comporta-se de maneira semelhante à Escreve Registro Randômicamente (operação nº 14). A diferença é que todo bloco de alocação requerido, antes de ser usado é preenchido com zeros.

ENTRADA: 1º byte : Código da operação (16);
2º byte : Identificador do registro de ativação do arquivo;
3º byte : Byte menos significativo da chave de acesso;
4º byte : Byte mais significativo da chave de

acesso;

5º ao 132º byte : Registro a ser gravado.

SAÍDA: 1º byte : Operação efetuada (03) ou;

Registro de ativação inexistente (255) ou

Busca após final físico do disco (254) ou

Diretório cheio (não conseguiu criar nova extensão) (253) ou

Erro de acesso ao disco (252) ou

Disco cheio (251) ou

Arquivo de escrita proibida (250).

18) **ALTERA ATRIBUTOS DE ARQUIVO**

Esta função permite alterar os atributos associados a um arquivo. Esses atributos são marcados nos bits de mais alta ordem dos bytes que formam o nome e tipo do arquivo. Os atributos seguem a convenção CP/M.

ENTRADA: 1º byte : Código da Operação (17);

2º byte : Unidade Lógica;

3º ao 10º byte : Nome do arquivo com novos atributos;

11º ao 13º byte : tipo do arquivo com novos atributos.

SAÍDA: 1º byte : Operação efetuada (03) ou

Arquivo não encontrado (255).

3.3 - Estrutura de Armazenamento no Servidor de Arquivo

3.3.1 - Armazenamento em Disco

O Armazenamento no Servidor de Arquivo é subdividido em quatro unidades de disco, sendo que cada unidade pode ser subdividida entre grupos de usuários. No protótipo temos apenas uma unidade (unidade A) e está dividida entre 16 grupos de usuários, que recebem os números de 0 a 15.

Cada unidade de disco é auto-contida em termo de armazenamento, ou seja, cada unidade tem estrutura de armazenamento totalmente independente, e isso simplifica o seu gerenciamento.

Examinaremos a seguir a estrutura lógica de uma unidade de disco. Abaixo temos a figura que representa logicamente a divisão do armazenamento em uma unidade.

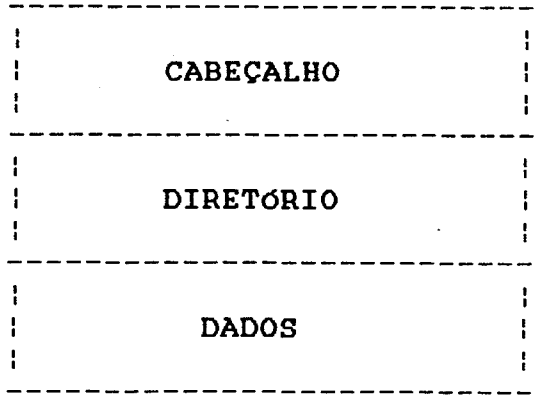

Figura 8 - Estrutura lógica de uma unidade de disco.

O disco para o SA é dividido em setores de 128 bytes. Rotinas de acesso ao disco devem mampear trilhas/setores reais em um

array de setores lógicos de 128 bytes cada.

Para facilitar o gerenciamento do disco, os setores são agrupados formando blocos de alocação. A escolha do tamanho do bloco de alocação, deve considerar tanto o tamanho da estrutura de controle de gerenciamento do espaço no disco, quanto a fragmentação interna, proveniente do uso de blocos. No protótipo, 8 setores contínuos (1 kbyte) formam um bloco de alocação.

O **gerenciamento** dos blocos de alocação é feito através de uma **mapa de alocação do disco**. Esse mapa tem um bit associado a cada bloco de alocação. Quando o bit está em 1, significa bloco alocado, em zero, bloco livre.

O **cabeçalho do disco** é composto apenas por um campo, o ponteiro para o início da lista de entradas do diretório livres.

O **diretório** está contido nos dois primeiros blocos de alocação (blocos 0 e 1). Ele é composto de **entradas de diretório**. O números de entradas de diretório a ser implantado (parâmetro no software do SA) dependerá da capacidade de armazenamento do disco, levando-se em conta a possível fragmentação de unidades físicas de armazenamento (trilhas e setores). No protótipo estão implementadas 64 entradas.

Cada **arquivo** presente no disco pode conter mais de uma entrada de diretório (à medida que o arquivo cresce), sendo as entradas pertencentes a um dado arquivo, encadeadas através de um ponteiro presente em um dos campos de cada entrada. Uma entrada do diretório é composta dos seguintes campos:

Estado : 1 bit. Representa arquivo aberto (bit=0), ou fechado (bit=1);

Número de Usuário : 1 byte. Contém o número do usuário. Se contiver valor hexadecimal E5, significa que a entrada está livre;

Nome do Arquivo : 8 bytes;

Tipo de Arquivo : 3 bytes;

Número da Entrada : 1 byte. Representa o número dessa entrada em relação as entradas alocadas a um arquivo. A primeira recebe valor zero;

Número de Registros : 1 byte. Representa o número de registros usados no último bloco de alocação pertencente a essa entrada;

Mapa de Alocação : 16 bytes. Contém 16 números de blocos de alocação do disco. Quando o número do bloco tiver o valor zero, significa nenhum bloco alocado.

Próxima Entrada : 1 byte : Se essa entrada estiver livre (Número de usuário igual a E5), esse campo aponta para a próxima entrada livre. Se essa entrada estiver alocada, esse campo aponta para a próxima entrada pertencente ao arquivo que a contém. Se o campo tiver valor zero, significa final de lista.

A parte de **dados do disco** é constituída dos blocos de alocação de número 2 ao máximo de blocos de alocação, no caso do protótipo número 310.

3.2.2 - Armazenamento em Memória

3.2.2.1 - Estruturas de Controle

As estruturas de controle necessárias ao gerenciamento de armazenamento no disco e ao processamento de arquivos são mantidas na memória. O **mapa de alocação do disco** e os **registros de ativação de arquivos** são as principais dessas estruturas. Os registros de ativação são um conjunto (organizado como uma lista ligada) de descritores de arquivo, alocados sob demanda quando da abertura de um arquivo, e liberados quando do fechamento. A idéia é manter na memória as informações sobre arquivos abertos, para evitar um fluxo de mensagens desnecessário entre a Estação de Trabalho e o Servidor de Arquivo. Assim, quando um cliente abre um arquivo, informações sobre o arquivo são copiadas em um registro de ativação e o número do registro de ativação usado é fornecido a ET. Esse número é o identificador daquele registro de ativação particular e terá que ser fornecido para os acessos naquele arquivo aberto. O número de registros de ativação é limitado (no protótipo são trinta).

Se em determinado instante todos os registros de ativação estiverem alocados, uma tentativa de abertura de arquivo será notificada com a mensagem de erro apropriada. É importante notar, que algum cliente, pode ter aberto um arquivo, contudo, esqueceu-se de fechá-lo. Alternativamente a operação ENCERRA_CESSAO pode ser usada para fechar todos os arquivos abertos numa determinada ET, e consequentemente, liberando todos os registros de ativação.

Um **Registro de Ativação** é composto pelos seguintes campos:

Diretório: 1 byte. Ponteiro para uma entrada no diretório;

Acesso: 1 byte. Contém o modo de acesso estabelecido para esse arquivo: Leitura_somente, Leitura_escrita, Multi_escrita, Leitura_exclusiva ou Excrita_exclusiva.

Registro_Atual: 1 byte. Número do registro dentro de uma entrada de diretório a ser acessado sequencialmente;

Houve_Gravação: 1 bit. Quando um arquivo é alterado esse "flag" é ligado para que no fechamento o diretório também seja alterado;

ET: 1 byte. Número da Estação de Trabalho que causou a abertura do arquivo;

Início: 1 byte. Ponteiro para a primeira entrada de diretório do arquivo;

Próximo: 1 byte. Aponta para o próximo registro de ativação livre. Quando esse registro de ativação estiver alocado, esse campo assume o valor zero.

Dada a possibilidade do acesso concorrente, podemos ter vários registros de ativação apontando para o mesmo arquivo no diretório.

Para melhor eficiência na manipulação do diretório, este é totalmente composto na memória, toda vez que se seleciona a unidade de disco correspondente. Somente quando necessário, o diretório no disco é acessado a fim de procederem-se alterações.

3.3.2.2 - **Cache**

O **Cache** foi implementado usando-se o Compilador Turbo-Pascal. O acesso ao disco é feito através das rotinas READBLOCK e WRITEBLOCK do Turbo-Pascal, cujos parâmetros são o número do setor e a quantidade de setores sequenciais a serem acessados. O número do setor é relativo ao início de um arquivo declarado no programa fonte. Na protótipo esse arquivo ocupa todo o disco e é estruturado como uma sequência de setores de 128 bytes. As rotinas READBLOCK e WRITEBLOCK poderiam ter sido implementadas usando-se as funções do BIOS (Basic Input/Output System) disponíveis numa memória ROM do IBM-PC, o que sem dúvida, as tornariam mais eficientes.

O Cache implementado tem a organização do tipo "**conjunto associativo**" e possue 60K bytes para dados que são divididos em 12 **conjuntos** com 5 **blocos** cada. O tamando do Cache foi limitado a 60K bytes pelo uso de estruturas (ARRAY) do Turbo-Pascal, porém, em uma implementação em "assembler" seu tamanho deve crescer para o máximo possível.

A seguir é descrito o Cache usando-se a linguagem PASCAL:

```
CONST    máximo_conjunto = 11;
         tamanho_conjunto = 5;
         tamanho_cache = 60;
```

```
TYPE    setor_lógico = array[1..128] of byte;
        bloco_cache = record
                        endereço : byte;
                        bloco : array[0..7] of
                                setor_lógico;
                      end;
        conjunto = array[0..4] of bloco_cache;

VAR     cache : array[0..máximo_conjunto] of conjunto;
```

A indexação, normalmente usada em Cache tipo "conjunto associativo", para se determinar o conjunto que contém uma dado bloco de alocação, foi simulada usando-se o módulo da divisão do endereço do bloco pelo número de conjuntos:

```
conjunto := número_bloco mod (máximo_conjunto + 1);
```

A busca do bloco endereçado em um dado conjunto, geralmente é feita através de memória associativa. No nosso caso é feita uma busca sequencial no conjunto (máximo de 5 comparações).

A política usada para acesso de escrita é do tipo "write-through", ou seja, toda vez que um bloco é alterado no Cache, essa alteração também é feita no disco.

A política usada para retirar blocos do Cache é do tipo LRU ("Least Recently Used"). Foi utilizado um algoritmo que é uma aproximação dessa política, e que simplifica a sua implementação. Esse algoritmo associa um bit para cada bloco do Cache. Toda vez

que o bloco é acessado, o bit é ligado. É feita então, uma busca circular nos blocos, e o primeiro bloco com o bit desligado é retirado. O bit de cada bloco é desligado quando o algoritmo o testa na tentativa de retirá-lo. A função implementada tem a seguinte estrutura:

```
FUNCTION lru : byte;
  begin
    repeat
      bit_lru[bloco_atual] := false;
      bloco_atual = (bloco_atual + 1) mod 5;
    until not bit_lru[bloco_atual];,
  end;
```

O Cache é acessado através da rotina acessa_cache. Assim, para a leitura se faz a chamada:

```
acessa_cache(leitura, bloco_alocação, deslocamento);
```

Para a escrita se faz:

```
acessa_cache(escrita, bloco_alocação, deslocamento);
```

3.4 - Protocolo de comunicação entre o Servidor de Arquivo e a Estação de Trabalho

A ligação entre o SA e uma ET é feita através de uma ligação ponto-a-ponto (par trançado), numa comunicação serial assíncrona, utilizando-se os circuitos 8251 da INTEL (na ET) e o INS8250 da NATIONAL (no SA) para controlar as saídas RS-232C.

O protocolo inplementaado é do tipo "stop-and-wait" e a unidade de transmissão ao nível de transporte é uma **mensagem** contendo um **pedido** para o SA, ou uma **resposta** para ET.

Em consequência da alta confiabilidade da ligação estabelecida, a ocorrência de erros é muito rara, e quando acontece, geralmente é decorrente de um fator não tratável ao nível do protocolo (ex: velocidade de transmissão diferente da recepção, queda de uma das estações, etc). Em face disso optou-se pela não retransmissão de mensagens, abortando-se a comunicação toda vez que se detecta um erro (vide Extensões - 4.0). O SA e a ET detectam um erro se acontece:

1- **Erro na recepção da transmissão.** Detectado através da programação do próprio circuito controlador (USART/ UART). Esses erros são: paridade, sobreposição de caracteres e erro de formato ("framing");

2- **Interrupção na comunicação.** Detectado através de "time-out".

Toda vez que a situação 1 ocorre no SA, este espera um determinado tempo para voltar a consultar a referida ET sobre novas mensagens. Isso força a ocorrência de "time-out" na ET, para que

esta possa também detectar a ocorrência do erro.

Ocorrendo um erro de comunição no SA (situação 1 ou 2), o controle volta ao início do protocolo a fim de processar a mensagem da próxima ET.

No SA o protocolo tem a seguinte estrutura:

```
REPITA Indefinidamente
 início
   PARA a próxima ET (obtida circularmente) FAÇA:
     SE existe algum pedido ENTÃO
       início
         CASO o pedido seja DE:
           operação válida:início
                              Envia sinal para a ET
                                    prosseguir;
                              Recebe o resto da mensagem.
                            fim
           operação inválida:início
                              Envia para ET o
                                   código de erro;
                              Aborta a comunicação;
                              Volta ao início do REPITA.
                            fim
         fim-caso
         Exucuta a operação no Servidor de Arquivo;
         Comunica resultado (acerto ou código de erro).
       fim
 fim.
```

Na Estação de Trabalho um módulo do sistema operacional prepara mensagem a ser transmitida (os formatos das mensagens aparecem em 3.2) e chama o protocolo. O protocolo na ET tem a seguinte estrutura:

início

Envia **pedido** ao Servidor de Arquivo;

Aguarda sinal de **prossiga** do SA; (o tempo associado ao "time-out" dura o suficiente para o SA atender a todas as ET's)

Envia o **Corpo da mensagem**;

Aguarda resultado do SA (**operação efetuada** ou **código de erro**);

Retorna ao sistema operacional.

fim.

Quando a comunicação é interrompida na ET, uma mensagem apropriada aparece na console e o controle volta ao sistema operacional (processador de comando de console - CCP).

3.5 - Hardware

O Servidor de Arquivo está atualmente implementado num microcomputador compatível com o IBM-PC (PC-XT ITAUTEC). O PC-XT da ITAUTEC é um micro de 16 bits montado a partir do processador 8088 da INTEL. Na sua versão original ele possui duas interfaces de comunicação serial no padrão RS232. Essas interfaces são controladas por dois circuitos INS8250 da NATIONAL SEMICONDUCTOR.

A rede montada para testar o SA é uma rede tipo estrela, onde o PC-XT está ligado a duas Estações de Trabalho, sendo que, o número de ET's pode ser expandido através de uma placa de comunicação serial, disponível no mercado. Cada placa possui 8 saídas seriais.

O PC-XT possui 2 "floppy-disks", um deles (320K bytes) é usado como disco do Servidor de Arquivo.

As ET's estão implementadas em um microcomputador de 8 bits (ITAUTEC-I7000), com processador NSC800 compatível com o Z80 da ZILOG. Cada ET tem pelo menos um "drive" de "floppy-disk" com capacidade de 320K bytes. O micro tem uma saída serial no padrão RS232c, controlada pelo circuito (USART) 8251 da INTEL.

A comunicação **serial assíncrona** entre o SA e a ET, é feita através de um **par trançado** contituído de três fios, que ligam uma saída serial do SA à saída serial da ET, e implementam os sinais: **transmite dados**, **recebe dados** e **terra** do padrão RS232.

3.6 - Software

3.6.1 - Software do Servidor de Arquivo

O Software do SA está escrito sobre o compilador Turbo-Pascal. As rotinas de manipulação das interfaces de comunicação serial, estão escritas em linguagem de máquina do INTEL 8088. O compilador Turbo-Pascal permite que se declare "procedures", dentro do programa fonte, escritas diretamente em linguagem de máquina, usando-se o código hexadecimal das instruções. O programa tem aproximadamente 2300 linhas de código fonte, 14 Kbytes de executável e 4Kbytes para dados. O software do Servidor de Arquivo tem a seguinte estrutura:

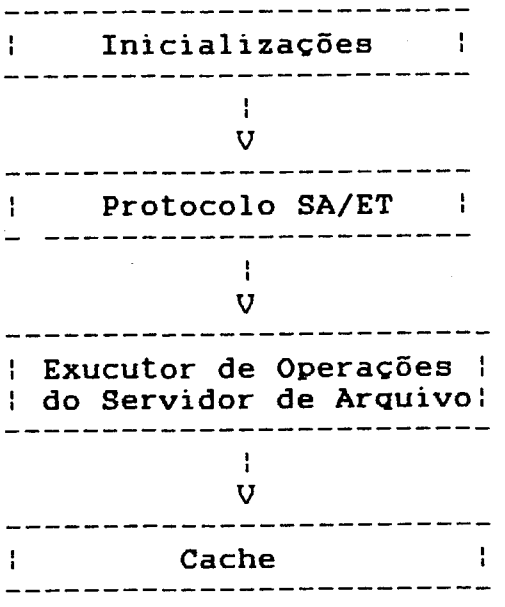

Figura 9 - Diagrama de Controle do Software do SA

O módulo de inicializações verifica se o disco foi anteriormente preparado para ser utilizado no SA (vide Utilitários -

3.7) e inicializa as estruturas de controle de armazenamento (registros de ativação, Mapa de alocação, etc). Também inicializa as interfaces de comunicação serial da seguinte forma:

Velocidade de transmissão : 9600 bps;

Paridade habilitada;

tipo de Paridade : Par;

Número de Stop Bits : Um;

Tamanho do caractere : 8 bits;

Limpa o "buffer" de recepção.

Por fim, o Cache é inicializado.

O módulo Protocolo SA/ET "varre" as Estações de Trabalho (interfaces Seriais) circularmente, executando uma mensagem de cada vez, para cada ET. O "time-out" associado as rotinas de recepção e transmissão de dados, é simulado usando-se instruções do Turbo-Pascal.

O Executor de Operações do SA executa a operação desejada, utilizando-se dos parâmetros armazenados no "buffer" de recepção do protocolo.

3.6.2 - Adaptação do Sistema Operacional Residente (CP/M) da Estação de Trabalho

O Cliente para o Servidor de Arquivo proposto é o sistema operacional da Estação de Trabalho (no caso, o CP/M - vide Apêndice). Programas de aplicação têm acesso ao Servidor de Arquivo através de chamadas ao sistema operacional. O sistema operacional residente (compatível com CP/M) na ET foi adaptado de forma a possibilitar o acesso ao SA. Também foi incluído o comando "transiente" KJOB, que executa a operação ENCERRA_CESSAO (operação 12) no SA.

A interface oferecida pelo SA é compatível com quase todas as chamadas do BDOS relativas ao sistema de arquivo. As excessões devem-se ao fato do SA ser compartilhado entre várias ET's, não permitindo a seus clientes a manipulação das estruturas internas do armazenamento. As chamadas do BDOS não implementadas geralmente são requeridas por utilitários CP/M (ex: STAT) para verificação de espaço no disco, ou mesmo para modificar a estrutura lógica de um disco. Em consequência, estes utilitários não são compatíveis com o sistema operacional, no que se refere ao SA. No capítulo 4.0 é sugerido a construção de utilitários que usam chamadas específicas do SA e que satisfaçam a necessidade de verificação de espaço em disco por parte do usuário.

A seguir são mencionadas as chamadas ao sistema operacional (BDOS) que não são encontradas na interface do cliente:

1) Chamada número 27: Obtém endereço do vetor de alocação.

O vetor de alocação é uma sequência de bits que informam quais blocos de alocação do disco estão sendo usados. Essa estrutura, no CP/M, é usada pelo BDOS para gerenciamento do espaço no disco. Também é utilizada por utilitários CP/M para determinar quanto de espaço está disponível no disco. Sendo o armazenamento do SA compartilhado entre várias ET's, a implementação dessa chamada perde o sentido.

2) Chamada número 31: Obtém endereço da tabela de parâmetros do disco (DPB).

Essa tabela descreve características do disco lógico especificado. A implementação dessa função não tem utilidade, já que o SA foi constituído de modo que a estrutura física de seus discos é transparente ao cliente.

O conhecimento do código fonte do sistema operacional, teria facilitado o processo de adaptação e permitiria domínio sobre estruturas como: "vetor de unidades read-only", "vetor de login","endereço de DMA", etc. Estruturas essas, usadas pelo sistema operacional para implementar diversas chamadas ao BDOS. Informações como endereço de DMA, áreas não usadas do BIOS, etc, foram descobertas no SIM/M (I7000-ITAUTEC) através do rastreamento e "decompilação" do sistema operacional.

Basicamente foram feitas as seguintes alterações:

1) Inclusão de um novo módulo no sistema operacional. Esse módulo foi denominado Servidor. O módulo Servidor foi escrito em **assembler** do Z80 com aproximadamente 800 li-

nhas de código fonte e ocupando aproximadamente 2K na memória principal. No disco ele recebe o nome de **Servidor.Sys**.

2) Inclusão de uma **rotina de carregamento** ("bootstrap") para carregar o módulo "Servidor.Sys".

O módulo "Servidor.Sys" está subdividido em dois módulos: **Novobdos** e **Protoc**. Com o sistema operacional da ET adaptado, chamadas ao sistema operacional são feitas ao módulo **Novobdos**. Ele é responsável por selecionar as chamadas feitas ao sistema operacional (BDOS), que são relativas ao processamento de arquivos no Servidor de Arquivo, e gerar num "buffer" parâmetros para que o módulo **Protoc** execute a referida operação no SA. O **Novobdos** tem a seguinte forma:

Caso **chamada ao sistema operacional** seja de:

Acesso ao **Servidor de Arquivo**: Início

Prepara "**buffer**" de **Mensagem**;

Executa o **Protocolo de Comunicação** (Protoc) com o SA.

Fim

Acesso ao Sistema de **disco local** ou

Outra operação qualquer: Chama **BDOS original**.

Fim_Caso.

O módulo **Protoc** é o protocolo de comunicação com o Servidor de Arquivo, e tem como finalidade básica enviar **pedidos** ao SA e **receber** resultados das operações efetuadas.

A comunicação com o Servidor de Arquivo é feita usando-se o circuito **8251 (USART)** da INTEL para controlar a saída RS232c.

O módulo **Protoc** tem a seguinte estrutura:

1) Inicializa o circuito **8251** da INTEL **(USART)**;

   Parâmetros de inicialização: Um stop bit;
   Paridade habilitada;
   Paridade Par;
   Caracteres de 8 bits;
   Fator de divisão de frequência igual a 16;
   Limpa "buffer" de recepção;

2) Inicializa o circuito **8253** da INTEL **(Temporizador)**;

   Parâmetros de inicialização: **Canal zero** - Programado para funcionar como contador de **"time-out"**;
   Canal um - Programado para gerar **"Baud rate"** da 8251.

3) Inicializa o **Sistema** de **Interrupções** da CPU **NSC800**;

   Parâmetros de inicialização: **Habilita** interrupção RST6.5 **(Temporizador - Canal zero)**;

Inibe interrupções de Teclado (RST5.5) e da USART-8251A (RST7.5).

Inicializa o endereço da rotina de tratamento de interrupção de "time-out".

4) Executa operação junto ao Servidor de Arquivo conforme o "buffer" de mensagens preenchido pelo módulo Novobdos.

3.7 - Utilitários

Dois utilitários foram desenvolvidos:

1) **FORMATA** - Tem como objetivo **inicializar o disco** para o formato requerido pelo Servidor de Arquivo. Basicamente esse utilitário gera a lista de entradas disponíveis e armazena o ponteiro do início dessa lista, no cabeçalho do disco.

2) **ALTERA** - tem por objetivo **consultar** e **alterar** entradas no diretório e também o cabeçalho do disco. Também pode ser utilizado para mostrar na tela o conteúdo ASCII dos blocos de alocação.

4.0 - EXTENSÕES

O Servidor de Arquivo implementado pode ser facilmente alterado para estabelecer quotas máximas, tanto de armazenamento permanente quanto temporário para cada usuário, que por sua vez, teria que ser cadastrado junto ao SA. O usuário de número zero poderia ser considerado como de acesso público. Assim, todos os programas de interesse geral (compiladores, editores, etc), poderiam estar disponíveis nessa área. Com essa alteração o comando **USER** também seria alterado, de modo a restringir o acesso a determinada área, ao fornecimento de uma senha por parte do usuário. Senha essa, estabelecida durante o cadastramento do usuário. O comando **KJOB** também teria que ser alterado de modo a desfazer a ligação estabelecida pelo comando USER.

Duas operações terão que ser adicionadas à interface do cliente. Uma que informe o espaço (real) de disco utilizado por dado arquivo, outra que informe a capacidade total do disco e o espaço utilizado. É importante notar que essas operações estão estreitamente comprometidas com a extensão proposta anteriormente, que atribui quotas de disco a usuários. Acrescentadas essas operações, deve-se também construir utilitários que permitam a visualização de tais informações por parte do usuário na console.

Na implementação de **Trancas**, para controle de concorrência a arquivos, quando se estabelece uma Tranca, o arquivo fica preso até que o usuário resolva liberá-lo através de uma operação de fechamento de arquivo. Dessa forma pode ocorrer uma situação em que uma aplicação fique indefinidamente impedida de ser executa-

da, pois o arquivo requerido por ela está preso para outro cliente. Para solucionar esse problema (LOCKOUT), pode-se associar às Trancas um limite de tempo (TIME-OUT), que seria considerado entre dois acessos ao mesmo arquivo e que, se expirado, causaria a quebra da Tranca, consequentemente possibilitando que um outro cliente ganhe o direito de acesso ao arquivo.

A implementação do protocolo de comunicação entre o SA e as ET's não considera a recuperação de erros de transmissão, apenas detecta o erro, e através de "TIME-OUT" aborta a aplicação. Considerando a possibilidade da permanência desse protótipo em ambientes mais hostis (ex: sala sem ar condicionado) e também considerando uma possível implementação do SA num ambiente multi-usuário, em que o SA compartilha "hardware" com outros processos, possibilitando o atraso no recebimento de mensagens (OVERRUN), torna-se necessária a implementação de recuperação de erros, que poderia ser feita simplesmente retransmitindo a mensagem uma quantidade fixa de vezes.

Uma outra extensão importante, seria a construção de um **processador de comando de console** (PCC) que incluísse os novos comandos USER e KJOB, e que considerasse o compartilhamento concorrente de arquivos. No PCC do sistema operacional adaptado, quando um arquivo é aberto para leitura (ex: O que ocorre com o comando TYPE), ele não é fechado. No SA isso causa problemas, pois todo arquivo aberto além de alocar um "registro de ativação", fica também protegido contra operações que alterem o arquivo. Na implementação do PCC aqui proposta, o comando TYPE ao final da operação fecharia o arquivo. No sistema atual, essas si-

tuações foram contornadas pela utilização do comando KJOB e pelo fato do SA fechar automaticamente arquivos sequenciais que chegaram ao último registro.

Para possibilitar Estações de Trabalho sem armazenamento de disco local, poderia ser incluída uma rotina de carregamento numa ROM na ET para carregar o sistema operacional a partir do próprio SA. No caso do microcomputador utilizado (I7000-ITAUTEC), isso poderia ser feito usando-se uma entrada para cartucho disponível no módulo base do micro.

O SA poderia também ser alterado de modo a dispor de um gerenciador de **fila de impressão de arquivos**. Os pedidos de impressão poderiam ser feitos pelas ET's utilizando-se um comando pertencente ao novo PCC.

Uma última sugestão seria a de rescrever Servidor de Arquivo na linguagem de programação C, escrevendo-se as rotinas que acessam diretamente os "drives" de disco usando-se as rotinas do "BIOS" disponíveis numa memória ROM do IBM-PC. Dessa forma poderia-se dispensar o uso do sistema operacional PC-DOS. Esse procedimento iria fazer com que o acesso ao disco fosse otimizado, bem como o sistema como um todo, já que seriam retiradas as restrições impostas pelo uso do compilador TURBO-PASCAL (ex: O não acesso a nível de bit).

5.0 - CONCLUSOES

O desenvolvimento do protótipo mostrou que é possível a utilização de um Servidor de Arquivo usando apenas o hardware de micro-computadores atualmente comercializados.

Também verificou-se que a ausência de mecanismos de transações atômicas não é uma restrição significativa, já que o sistema implementado comporta-se de maneira semelhante ao CP/M, cuja compatibilidade foi um dos nossos objetivos.

Apesar da tentativa de melhorar a performance do SA, através da implementação de um **Cache**, constatamos que, não obstante a melhora obtida (cerca de 10%), o gargalo do sistema é a **velocidade de comunicação**, que no nosso caso é de 9600 bps. Assim, espera-se que numa implementação de comunicação mais rápida (acima de 50K), a performance geral melhore significativamente.

6.0 - Bibliografia

6.1 - Servidores de Arquivo:

[BIRRELL 80]

BIRRELL, A. D. & NEEDHAM, R. M.

A Universal File Server,IEEE Trans. Soft. Eng.,

SE-6,5 (Setembro 1980),450-453.

[DELLAR 82]

DELLAR, C. **A File Server for a Network of Low Cost Personal Microcomputers**. Softw. Pract. Exper. 12, 1051-1068.

[DION 80]

DION, J. **The Cambridge File Server**

Oper. Syst. Rev., 14,4 (Outubro 1980), 26-35.

[FRIDRICH 81]

FRIDRICH, M. & OLDER, W. **The Felix File Server**,

Proc. of the Eighth Symposium on Operating Systems Principles, Asilomar, California, (Dezembro 1981). pp. 37-44.

[GARNETT 80]

GARNETT, N. H. & NEEDHAM, R. M.

An Asynchronous Garbage Collector for The Cambridge File Server. ACM SIGOPS Oper. Syst. Rev., 14,4(Outubro 1980),36-40.

[MACEDO 86]

MACEDO, Raimundo J. de A.

Servidores de Arquivo: Uma introdução

ANAIS da Primeira Semana de Informática da UFBa, Março de 1986.

[MITCHELL 82]

MITCHELL, J. G. e DION, J. **A Comparison of Two Network-based File Servers**, CACM,25(4),Abril 1982.

[MITCHELL 83]

MITCHELL, J. G. **File Servers**, Lecture Notes in Computer Science, Vol. 184, pags. 221-259. Springer-Verlag (1983).

[NEEDHAM 79]

NEEDHAM, R. M. **Adding Capability Access to Conventional File Servers**, Oper. Syst. Rev. 13(1),3-4(1979).

[PAXTON 79]

PAXTON, W. H. **A Client-Based Transaction System to maintain data integrity**. In Proc. of Seventh Symp. on Op. Systems principles. Asilomar, California, Dezembro 1979, pp. 18-23.

[STURGIS 80]

STURGIS, H. E., MITCHELL, J. G. e ISRAEL, J. **Issues in the Design and Use of a Distributed File System**, SIGOPS Op Sys Rev, 14(3), Julho 1980, pp. 55-69.

[SVOBODOVA 84]

SVOBODOVA, L. **File Servers for Network-Based Distributed Systems**, Computing Surveys, vol. 16 Num. 4, Dezembro 1984.

[SWINEHART 79]

SWINEHART, D., McDANIEL, G. e BOGGS, D. R.

WFS: A Simple Shared File System for a Distributed Environment, Proc. of Seventh Symposium on Operating Systems Principles, Asilomar, California, Dezembro 1979, pp. 9-17.

[TOLEDO 86]

TOLEDO, Maria Beatriz F. de

Desenvolvimento de um Servidor de Arquivo com Transações Atômicas, Tese de Mestrado, IMECC-UNICAMP, Abril de 1986.

6.2 - CP/M (Control Program/Monitor)

[BARBIER 84]

BARBIER, Ken.

CP/M Techniques, New Jersey, USA, Prentice-Hall, 1984.

[FERNANDEZ 84]

FERNANDEZ, Judin & ASHLEY, Ruth.

Usando CP/M, Um Guia em Ensino Programado,

Ed. Campos (1984).

[GOLDEN 86]

GOLDEN, Donald & PECHURA, Michael.

The Structure of Microcomputer File Systems,

Comunications of the ACM - Vol. 29, Nº 3, Março 1986.

[HOGAN 83]

HOGAN, Thom.

CP/M Guia do Usuário

Ed. McGRAW-HILL do Brasil LTDA (1983).

[JOHNSON-LAIRD 83]

JOHNSON-LAIRD, Andy.

The Programmer's CP/M Handbook,

Osborne/Mcgraw-Hill, Berkley, California, USA (1983).

6.3 - Bibliografia Complementar

[CHANEY 84]

CHANEY, Roy & JOHNSON, Brian.

Maximizing Hard-Disk Performance

How Cache Memory can dramatically affect Transfer Rate,

Byte, Vol. 9, Nº 5, Maio 1984.

[McNAMARA 82]

McNAMARA, John E.

Technical Aspects of Data Comunications,

2a. edição, USA - Digital Press, 1982.

[POHM 81]

POHM, A. V. & SMAY, T. A. (IOWA STATE UNIVERSITY).

Computer Memory Systems,

Computer - Outubro 1981.

[SALTZER 75]

SALTZER, J. H. e SCHROEDER, M. D.

The Protection of Information in Computer Systems,

Proc. of IEEE, vol. 63, Num. 9, Setembro 1975.

[SMITH 82]

SMITH, Alan J. (UNIVERSITY of CALIFORNIA, BERKLEY).

Cache Memories,

Computing Surveys, vol. 14, Nº 3 - Setembro 1982.

[STRECKER 78]

STRECKER, William D.

Cache Memories for PDP-11 Family Computers (cap. 10).

BELL, C. Gordon; MUDGE J. Croig; McNAMARA, John E.

Computer Engineering 1a. edição USA - Digital Press, 1978 - pags. 263-267.

[TANENBAUM 81]

TANENBAUM, Andrew S.

Computer Networks,

Prentice-Hall, USA (1981).

7.0 - APENDICE : Características do CP/M Relevantes à implementação do Servidor de Arquivo

7.1 - Estrutura do CP/M

O CP/M (Control Program/Monitor) é composto de três módulos: O Processador de Comando de Console (CCP), o Sistema Operacional Básico de Disco (BDOS) e o Sistema Básico de Entrada e Saída (BIOS).

O CCP tem como finalidade interpretar/executar comandos digitados na console.

O BDOS é o módulo central do CP/M, ele controla todos os recursos do micro-computador (discos impressora, vídeo, etc.).

Os dispositivos para o BDOS são associados a nomes lógicos (ex: impressora recebe o nome lógico "LST:"). Dispositivos lógicos podem ser associados a dispositivos físicos através de uma chamada ao sistema operacional (BDOS).

O BDOS considera a existência de até 16 discos conectados ao sistema, que são numerados de 0 a 15, e denominados "drive" A, "drive" B, "drive" C, até "drive" P.

O BIOS é formado por um conjunto de rotinas que acessam diretamente os dispositivos físicos (discos, teclados, ect.) e que são utilizadas pelo BDOS.

Na figura a seguir temos a estrutura do CP/M em termos de camadas.

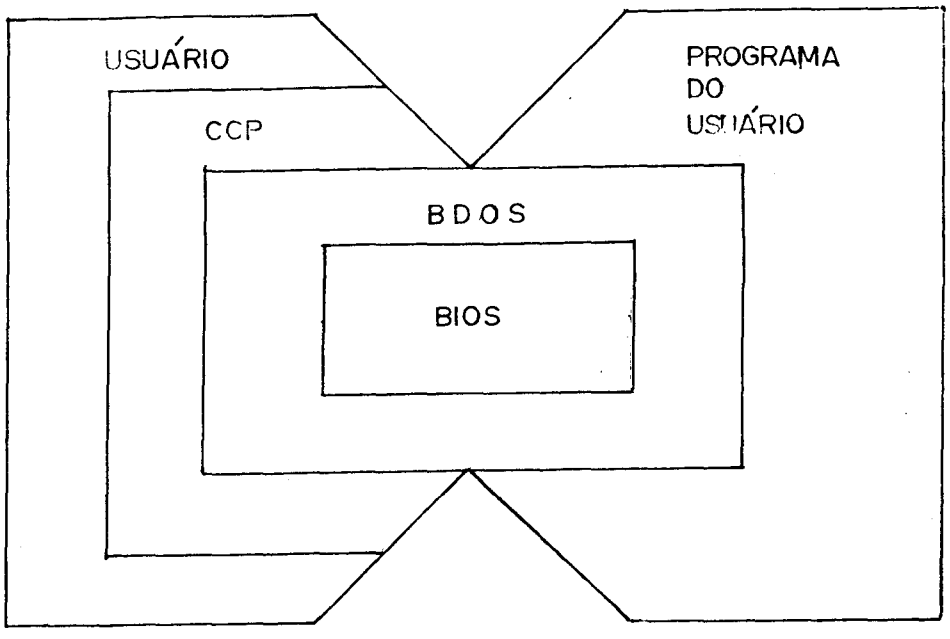

Figura 10 - Estrutura do CP/M

Quando o CP/M está carregado na memória, os primeiros 256 bytes formam a **Página Básica**, onde são armazenados parâmetros para o CP/M. Os mais importantes são:

1º ao 3º byte: Instrução de desvio para o BIOS (rotina de "warmboot");

4º byte: Serve para fazer a associação entre dispositivos lógicos e físicos.

5º byte: Os 4 bits mais significativos indicam o usuário corrente (0 a 15). Os bits menos significativos indicam o "drive" corrente (A a P).

6º ao 8º byte: Instrução de desvio para o BDOS. Chamadas ao

Sistema operacional são feitas através de uma instrução de desvio para o BDOS. Ou seja, o programa de aplicação usa a instrução "CALL 5". Os parâmetros relativos ás chamadas são passados em registradores da Unidade Central de Processamento.

A figura a seguir mostra a disposição do CP/M padrão [Johnson-Laird 83] carregado na memória.

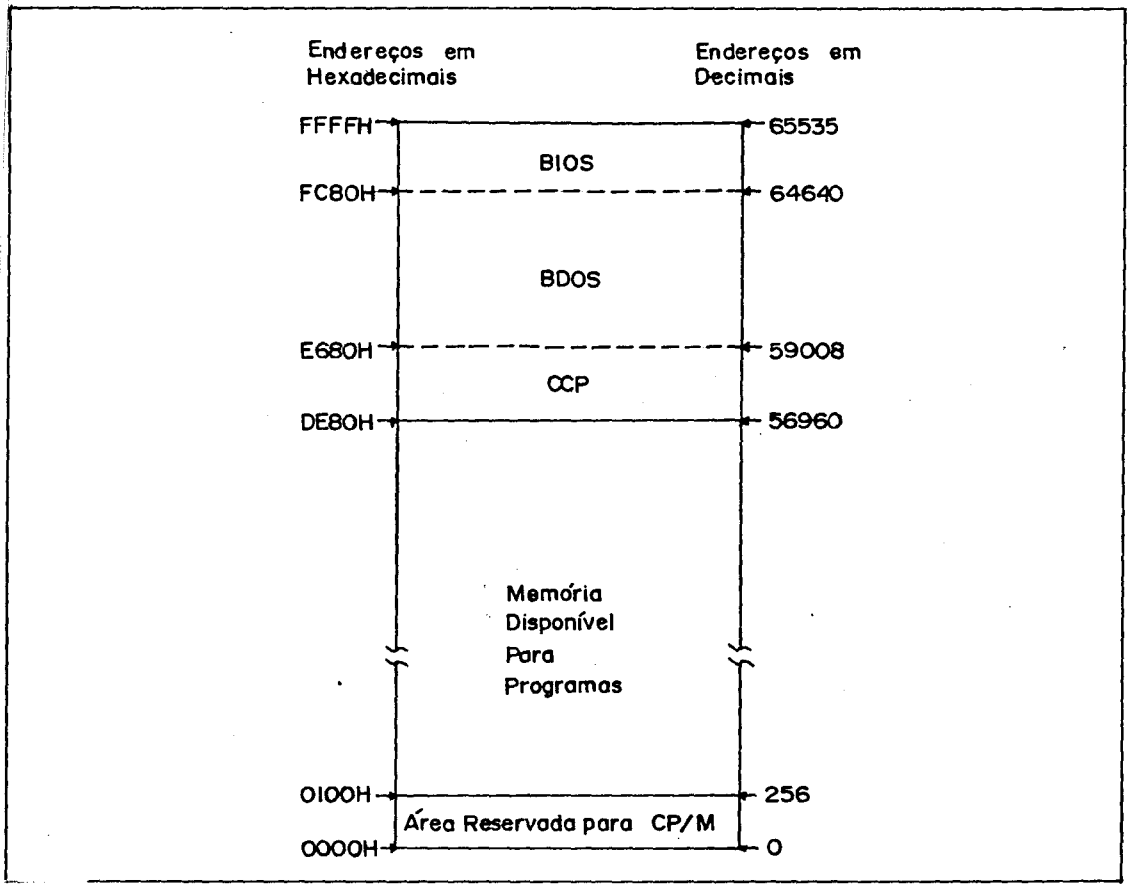

Figura 11 - Disposição do CP/M na memória

7.2 - Sistema de Arquivo CP/M

O BDOS é o módulo encarregado do gerenciamento do sistema de arquivo. Um disco CP/M é subdividido em três áreas: **Área reservada** para armazenamento do próprio CP/M, O **diretório de arquivos** e a área destinada ao **armazenamento de dados.**

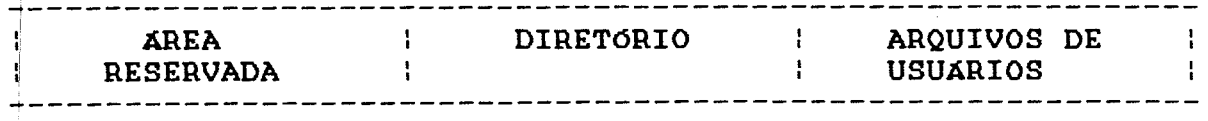

Figura 12 - Disco CP/M

Para o BDOS, um disco CP/M é uma sequência de setores lógicos de 128 bytes cada. A fim de facilitar o gerenciamento do armazenamento, o CP/M agrupa alguns setores contínuos para formar um bloco de alocação (no CP/M padrão 8 setores formam um bloco). Dessa forma, o disco é então uma sequência de blocos de alocação. No CP/M padrão os blocos 0 e 1 são reservados ao diretório, os demais ao armazenamento de dados.

Cada entrada do diretório possue 32 bytes e tem o seguinte formato:

1º byte: **Número do Usuário.** Um disco CP/M é partilhado entre vários grupos de usuários. Quando o Número de Usuário receber o valor hexadecimal "E5", signifi-

ca que esta entrada de diretório está livre.

2º ao 9º byte: **Nome do Arquivo.**

10º ao 12º byte: **Extensão do Nome do Arquivo** (Tipo). Os bits de mais alta ordem do tipo são usados como "flags" para o sistema de arquivo (ex: o bit de mais alta ordem o 10º byte, com valor 1, significa arquivo protegido contra escrita).

13º byte: "**Extent**". Esse campo é utilizado se um arquivo necessita possuir mais que uma entrada no diretório. Ou seja, se o arquivo não comporta nos blocos de alocação destinados a uma entrada de diretório. Quando um arquivo ocupa mais que uma entrada de diretório. Cada entrada terá nome, extensão e número de usuário iguais, sendo diferenciadas pelo número do "Extent", que recebe valor zero para a primeira entrada alocada, valor um para a segunda, etc.

14º ao 15º byte: Reservado para uso do CP/M.

16º byte: **Números de Setores** alocados no último bloco de alocação dessa entrada.

17º ao 32º byte: **Mapa de Alocação do Disco**. O Mapa de Alocação do Disco contém 16 números de blocos de Alocação.

Para que um usuário possa acessar determinado arquivo no disco, ele tem que usar, o que se chama, **Bloco de Controle de Ar-**

quivo. As informações declaradas pelo usuário no Bloco de Controle de Arquivo, incluem: O número do "drive" onde reside o arquivo, o nome do arquivo, a extensão do arquivo e o número de "extent".