UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE GERENCIADOR DE ARQUIVOS PARA AMBIENTE DISTRIBUÍDO

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

CATIA SILENE POSSAMAI

BLUMENAU, FEVEREIRO/2000

1999/2-95

PROTÓTIPO DE GERENCIADOR DE ARQUIVOS PARA AMBIENTE DISTRIBUÍDO

CATIA SILENE POSSAMAI

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Antônio Carlos Tavares — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Antonio Carlos Tavares

_____

Prof. Sérgio Stringari

_____

Prof. Maurício Capobianco Lopes

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

RESUMO

Este trabalho visa o desenvolvimento de um protótipo de software para tratar o problema de recuperação e organização de arquivos em um ambiente distribuído. O protótipo constitui-se em um gerenciador de arquivos que torne transparente ao usuário a localização física dos arquivos que deseja armazenar e recuperar, e permite a organização dos arquivos em estruturas de diretórios. Serão estudados e implementados os serviços de arquivos e diretórios fornecidos por um sistema de arquivos de um sistema operacional distribuído.

ABSTRACT

This work develops the software prototype to treat the recovery problem and organization of files in a distributed environment. The prototype constitutes a files manager that turns transparent to the user the physical location of the files that wants to store and to recover, and it allows the organization of the files in directories structures. They will be studied and implemented the services of files and directories supplied by a files system of a distributed operating system.

1 INTRODUÇÃO

Inovações na capacidade de processamento de comunicação propiciaram grande desenvolvimento das redes de computadores. Com isso, foi adicionado ao processamento de dados o custo de telecomunicações que, com o tempo, tornou-se parcela considerável.

Neste contexto, no fim dos anos 70, surgiram os sistemas distribuídos, integrando os recursos computacionais e de telecomunicações, de forma a reduzir os custos de transmissão de dados e de colocar a capacidade de processamento junto ao usuário da aplicação [STR84].

Sistemas distribuídos são sistemas de computação compostos por um certo número de processadores conectados através de rede. Estes sistemas são uma evolução dos sistemas fortemente acoplados, onde uma aplicação pode ser executada por qualquer processador. Os sistemas distribuídos permitem que uma aplicação seja dividida em diferentes partes (aplicações distribuídas), que se comunicam através de linhas de comunicação, podendo cada parte ser processada em ambiente independente [MAC97].

Os sistemas distribuídos podem trazer muitas vantagens, sendo que a principal delas, sem dúvida, é o compartilhamento de recursos (hardware, software, dados, etc.). Cada componente do sistema também pode possuir seu próprio sistema operacional, memória, processador e dispositivos. Para os usuários e seus aplicativos, é como se não existisse uma rede de computadores, mas um único sistema centralizado, de forma a tornar mais acessíveis os recursos disponíveis [MAC97] e [KIR88].

As operações realizadas em sistemas distribuídos normalmente são mais complexas de serem implementadas, por terem que tratar a questão de distribuição física e lógica, segurança de informações, controle de acesso, e garantir uma certa abstração aos usuários quanto às operações internas necessárias para manter os ítens abordados acima em funcionamento.

O armazenamento e recuperação de informações são atividades essenciais para qualquer tipo de aplicação. Para isso, os sistemas operacionais implementam arquivos, permitindo realizar operações de arquivamento e recuperação das informações.

Os arquivos são unidades com informações (programas ou dados) armazenados em discos ou em qualquer outro dispositivo externo de armazenamento. Os arquivos são gerenciados pelo sistema operacional de modo a facilitar o acesso dos usuários ao seu conteúdo. A parte do sistema operacional responsável por essa gerência é denominada de sistema de arquivos [MAC97] e [TAN95].

Em sistemas operacionais distribuídos, a gerência de arquivos exige o tratamento de questões importantes, como aspectos de transparência na localização e distribuição física dos arquivos, compartilhamento entre os diversos componentes do sistema e forma de organizar os arquivos entre os diversos usuários. Para que isso se torne possível, existem técnicas que um sistema de arquivos deve implementar para garantir o funcionamento adequado de suas operações.

1.1 OBJETIVOS

O objetivo principal do trabalho é desenvolver um protótipo de gerenciador de arquivos, que permita armazenar, recuperar e organizar dados em um ambiente distribuído, deixando transparente ao usuário a localização física dos arquivos, tanto nas operações de arquivamento quanto na recuperação dos arquivos.

Os objetivos secundários do trabalho são:

a) estudar e descrever aspectos da estrutura de um sistema de arquivos distribuídos, modelos de acesso aos arquivos, organização dos arquivos dentro do sistema, estruturas de diretórios, transparência no acesso e armazenamento, identificação de arquivos, semânticas de compartilhamento, formas de implementar a segurança dos dados e performance do sistema de arquivos;
b) estudar e descrever estudos de casos referentes à sistemas de arquivos distribuídos encontrados na literatura;

1.2 ORIGEM DO TRABALHO

A motivação para estudar e demonstrar, através de um protótipo, os sistemas de arquivos distribuídos, surgiu, em primeiro lugar, do interesse pessoal pelo assunto, e pela observação da grande carência no entendimento, por parte dos acadêmicos, em relação ao

assunto de sistemas operacionais. Idealizou-se que, o fato de estudar e descrever uma importante parte que compõem os sistemas operacionais, possa auxiliar, através de um exemplo prático, a pesquisa e compreensão nesta área.

1.3 ESTRUTURA DO TRABALHO

O trabalho está organizado da seguinte forma:

O capítulo 1 traz uma introdução ao tema do trabalho, origem e objetivos a serem alcançados com o desenvolvimento do mesmo;

O capítulo 2 define conceitos básicos em sistemas operacionais, necessários para o entendimento dos demais tópicos. Este capítulo retrata também os sistemas de arquivos, suas estruturas e funções;

O capítulo 3 explora o assunto de sistemas de arquivos distribuídos: nomeação de arquivos, questões de transparência, compartilhamento de arquivos, como solucionar problemas decorrentes do compartilhamento de arquivos, replicação de arquivos e outros temas referentes à sistemas de arquivos distribuídos;

No capítulo 4 é feita uma explanação de três estudos de casos: o sistema de arquivos *Andrew*, o *Network File System (NFS)* e o *Amoeba*;

No capítulo 5 está a descrição de um protótipo para gerenciador de arquivos distribuídos, a especificação das operações implementadas e da funcionalidade do protótipo;

O capítulo 6 demonstra o protótipo em funcionamento, uma aplicação que utiliza o gerenciador de arquivos desenvolvido no trabalho. Este capítulo demonstra também os testes realizados e os resultados obtidos, defrontando com os objetivos iniciais do trabalho;

O capítulo 7 traz as considerações finais sobre o trabalho e sugestões para trabalhos futuros que podem utilizar-se deste trabalho como base e início de pesquisa;

O capítulo 8 traz as referências bibliográficas utilizadas como pesquisa no desenvolvimento do trabalho;

O capítulo 9 contém anexos do código fonte do protótipo.

2 FUNDAMENTAÇÃO TEÓRICA

Um sistema de computação possui diversos componentes de hardware e software. Ao utilizar estes componentes, a maioria dos usuários não precisa se preocupar com a maneira como é realizada a comunicação entre os componentes, ou com os inúmeros detalhes envolvidos no processamento. Deste conjunto de dispositivos, o software de sistema operacional é quem oferece a interface entre o usuário e os recursos disponíveis no sistema. O sistema operacional possui um conjunto de rotinas específicas que interpretam as solicitações do usuário e acionam os componentes de hardware envolvidos. Dessa forma, torna-se transparente a comunicação entre dispositivos do sistema e usuários, facilitando as operações realizadas em computador.

2.1 CONCEITOS BÁSICOS

Um sistema operacional é um conjunto de rotinas executado pelo processador, cuja função é controlar o funcionamento do computador, como um agente dos diversos recursos disponíveis no sistema. O sistema operacional serve de interface entre o usuário e os diversos componentes e recursos de um sistema de computação, tornando a comunicação entre ambos transparente e permitindo ao usuário um trabalho mais eficiente e com menos chances de erros. Existem diversos tipos de sistemas operacionais, que controlam diferentes modelos de organização de hardware. Estes tipos estão relatados a seguir.

2.1.1 SISTEMAS OPERACIONAIS MONOPROGRAMÁVEIS /MONOTAREFA

São sistemas operacionais que se caracterizam por executar uma única tarefa/programa de cada vez. Qualquer outro programa, para ser executado deve esperar o término do programa corrente. Os sistemas monoprogramáveis fazem com que o processador, memória e periféricos se voltem exclusivamente para a execução de uma única tarefa.

2.1.2 SISTEMAS OPERACIONAIS MULTIPROGRAMÁVEIS /MULTITAREFA

Neste tipo de sistema operacional, o diversos recursos disponíveis podem ser utilizados por vários programas, havendo um compartilhamento de processador, memória, periféricos e dados. Os sistemas operacionais multiprogramáveis preocupam-se em controlar o acesso concorrente aos recursos, entre os diversos programas. Dessa forma, o sistema faz com que o processador possa utilizar o tempo ocioso de algum programa para realizar outra operação, havendo então um aumento de produtividade dos usuários e redução nos custos de processamento.

2.1.3 SISTEMAS DE COMPUTAÇÃO COM MÚLTIPLOS PROCESSADORES

Estes sistemas caracterizam-se por possuir dois ou mais processadores interligados, trabalhando em conjunto. Os sistemas operacionais com múltiplos processadores controlam a comunicação entre os processadores e o compartilhamento de memória e dispositivos de entrada e saída. Estes sistemas classificam-se em fracamente acoplados e fortemente acoplados.

Os **sistemas fortemente acoplados** possuem dois ou mais processadores, compartilham uma única memória e são controlados por apenas um único sistema operacional.

Os **sistemas fracamente acoplados** caracterizam-se por possuir dois ou mais sistemas de computação, interligados por linhas de comunicação. Cada sistema funciona de forma independente, possuindo seu(s) próprio(s) processador(es), memória e dispositivos de entrada e saída.

Os sistemas operacionais possuem um conjunto de rotinas que oferecem serviços aos usuários do sistema e suas aplicações. Este conjunto de rotinas é chamado de **núcleo do sistema operacional** ou ***kernel***. As principais funções do núcleo são:

a) tratamento de interrupções;
b) criação e eliminação de processos;
c) sincronização e comunicação entre processos;

d) escalonamento e controle de processos;
e) gerência de memória;
f) operações de entrada e saída;
g) gerência do sistema de arquivos;
h) contabilização e segurança do sistema.

2.2 SISTEMAS OPERACIONAIS CENTRALIZADOS E DISTRIBUÍDOS

Os sistemas operacionais percorreram um longo caminho desde os anos 50. Originalmente foram projetados para controlar um único computador, gerenciando um só processador. Estes sistemas operacionais podem ser chamados de sistemas operacionais centralizados.

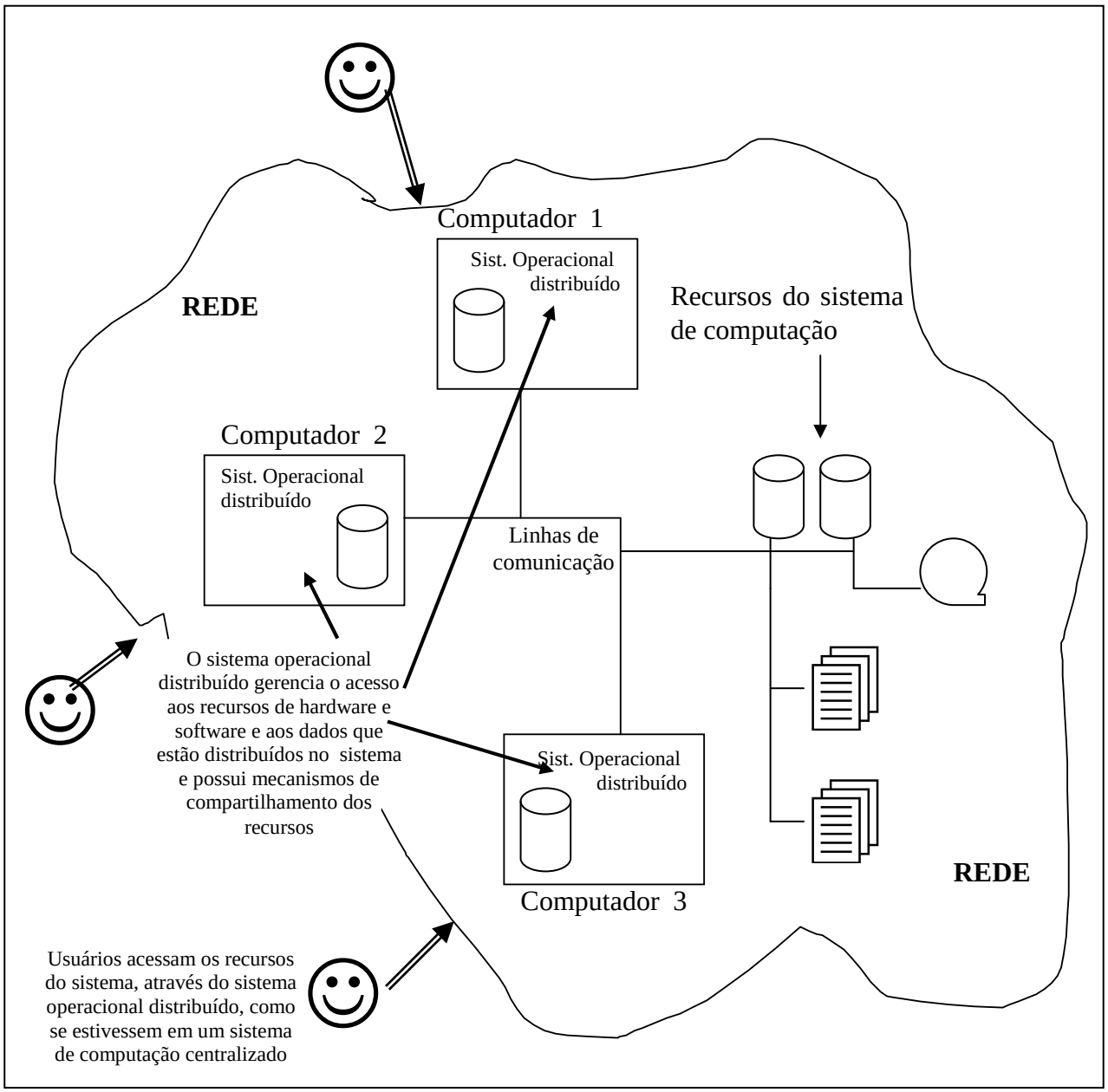
Os novos sistemas operacionais são desenvolvidos para executar em uma série de máquinas conectadas através de rede, sendo denominados sistemas operacionais distribuídos. Os sistemas operacionais distribuídos surgiram devido ao grande avanço da tecnologia. O desenvolvimento dos microprocessadores, que ganham um poder de processamento cada vez maior, teve considerável influência no aparecimento desses sistemas. Outro fator importante foi o surgimento e rápido crescimento das redes locais de alta velocidade, que vieram permitir a conexão de muitos computadores, compartilhando e transferindo um volume cada vez maior de informações. O resultado dessas duas tecnologias permite criar facilmente sistemas de computação com grande número de processadores ligados através de redes de alta velocidade. Estes sistemas são denominados sistemas distribuídos [TAN95].

Nos sistemas distribuídos, uma aplicação pode ser dividida em diferentes partes, podendo estas partes serem executadas por diferentes processadores, em ambientes independentes, comunicando-se através de linhas de comunicação [MAC97].

Um sistema operacional distribuído é um sistema operacional que opera em ambientes distribuídos, oferecendo mecanismos de compartilhamento de recursos e aplicações. O sistema gerencia o acesso e recuperação de dados distribuídos na rede, fornecendo formas de proteção e controle. Além disso, os sistemas operacionais distribuídos preocupam-se com a transparência nas operações realizadas em ambientes distribuídos, de forma que os usuários

tenham a impressão de estarem trabalhando em um sistema centralizado, eliminando a preocupação com a localização física dos recursos de hardware e software (fig. 2.1)

FIGURA 2.1 – SISTEMA OPERACIONAL DISTRIBUÍDO



2.3 SISTEMA DE ARQUIVOS

As operações realizadas em computador envolvem programas e aplicações. Os programas em execução e toda a estrutura responsável para manter informações necessárias à sua execução são chamados de processos.

Qualquer aplicação realizada através de computador precisa armazenar, recuperar dados e muitas vezes compartilhá-los com outros processos. O modo como os usuários precisam lidar com estas tarefas depende de como o sistema operacional organiza e mantém estas informações nos meios de armazenamento, mediante a implementação de arquivos.

Os arquivos são unidades que contém informações logicamente relacionadas, podendo constituir-se de programas ou dados. Os programas contém instruções compreendidas pelo computador, e os arquivos de dados podem conter qualquer tipo de informações, como textos, registros de um banco de dados, etc.

A parte do sistema operacional responsável pelo gerenciamento e manutenção dos arquivos é denominada de sistema de arquivos. Através do sistema de arquivos, tarefas de recuperação e acesso ao conteúdo dos arquivos são disponibilizadas ao usuário, que ficam dispensados dos detalhes de como estes arquivos estão organizados nos meios de armazenamento, como é o controle de memória disponível e outros detalhes da implementação. Para os usuários, o aspecto a ser considerado é como um arquivo será identificado, como será protegido e quais as operações que podem ser realizadas sobre ele.

O sistema de arquivos controla a forma como os arquivos são organizados, como será o acesso aos dados, como serão identificados pelo usuário, quais os atributos e operações possíveis sobre eles.

É através do sistema de arquivos que os usuários terão uma interface para armazenar e recuperar seus dados, de forma transparente quanto aos detalhes de implementação e organização. E é através dele também que os diferentes processos do sistema poderão executar tarefas sobre os arquivos ou compartilhá-los com outros processos [TAN95] e [MAC97].

O sistema de arquivos pode oferecer serviços para atender as necessidades de armazenamento, controle e recuperação de arquivos e também serviços para a organização desses arquivos em diretórios.

2.3.1 SERVIÇO DE ARQUIVOS

O modo como o serviço de arquivos está estruturado depende de como o sistema representa os arquivos. Para alguns sistemas, um arquivo pode ser uma seqüência qualquer de bytes, como por exemplo no Unix e MS-DOS. Neste caso, o conteúdo e estrutura interna do arquivo fica por conta dos programas de aplicação.

Um arquivo pode também ser estruturado como uma seqüência de registros, onde sua localização é feita através de sua posição (número do registro) ou pelo valor de algum de seus campos, e chamadas ao sistema são feitas para leitura e escrita em um registro específico.

Um arquivo pode conter atributos, que são informações sobre o arquivo, mas que não fazem parte do seu conteúdo, por exemplo, nome, data de criação, direitos de acesso, tamanho.

Os arquivos podem possuir, também, proteções para controle de acesso ao seu conteúdo ou às operações referentes aos arquivos. Os arquivos podem ser protegidos por listas de capacidade ou listas de controle de acesso.

Nas **listas de capacidades**, o usuário recebe uma "capacidade" para os arquivos aos quais tem acesso. Esta capacidade representa quais as operações permitidas para o usuário, como somente leitura, ou leitura e escrita.

Nas **listas de controle de acesso**, cada arquivo tem associado uma lista de usuários que poderão acessar os arquivos, além do tipo de acesso permitido a cada um.

2.3.2 DIRETÓRIOS

Diretórios são estruturas de dados que contém entradas associadas aos arquivos onde são armazenadas informações como: localização física, nome, organização e demais atributos. Quando é solicitada a leitura de um arquivo, o sistema operacional procura sua entrada na estrutura de diretórios.

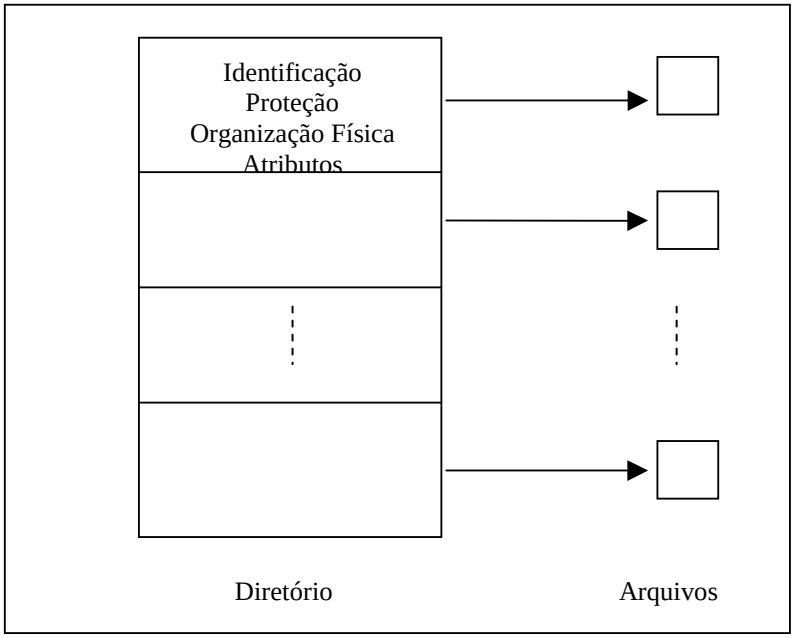
O sistema de arquivos pode oferecer o serviço de diretórios. O serviço de diretórios disponibiliza operações para criação e remoção de diretórios, identificação e mudança de nomes de arquivos e operações para movimentar arquivos de um diretório para outro.

O modo como o sistema de arquivos organiza os arquivos no disco determina a estrutura de diretórios. Os diretórios podem ser estruturados de diversas formas: estrutura de diretório de nível único, estrutura de diretórios de dois níveis e estrutura de diretórios de múltiplos níveis.

2.3.2.1 ESTRUTURA DE DIRETÓRIOS DE NÍVEL ÚNICO

Esta estrutura é a mais simples de ser implementada, mas não é a melhor maneira de organizar os arquivos em um disco. Ela consiste em um único diretório contendo todos os arquivos, conforme ilustrado na fig. 2.2. O usuário não pode criar nomes duplicados de arquivos nem pode separar seus arquivos dos demais arquivos do disco.

FIGURA 2.2 - ESTRUTURA DE DIRETÓRIOS DE NÍVEL ÚNICO

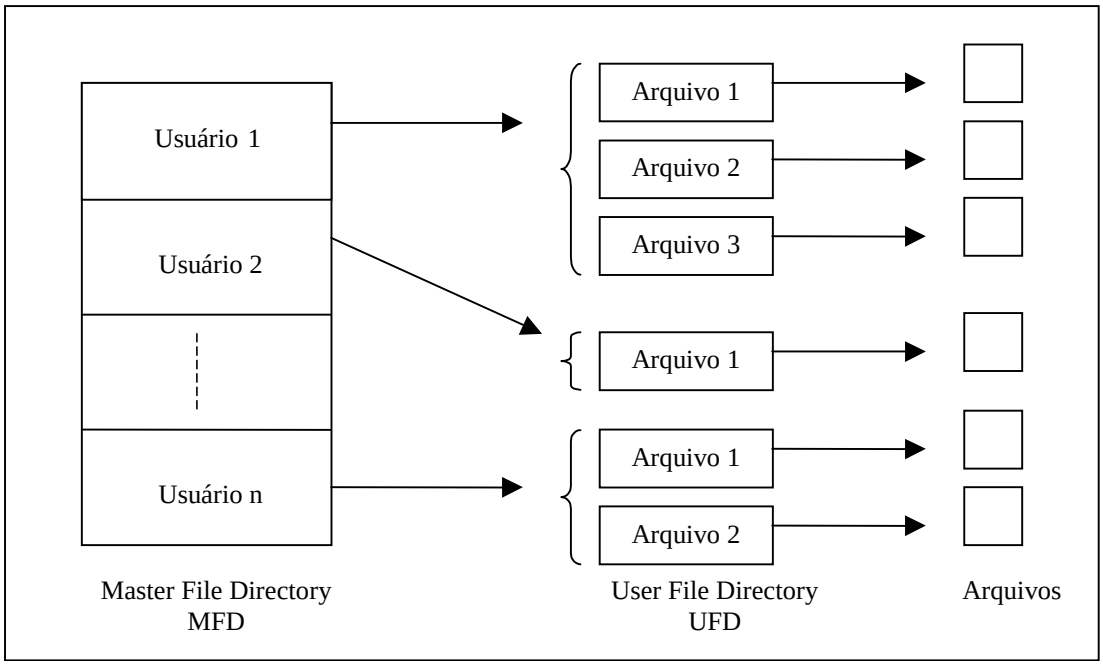


FONTE: [MAC97]

2.3.2.2 ESTRUTURA DE DIRETÓRIOS DE DOIS NÍVEIS

Pela limitação da estrutura de nível único, criou-se também a estrutura de dois níveis. É mantido um diretório principal, denominado *Master File Directory* (MFD), onde cada usuário possui um diretório individual, denominado *User File Directory* (UFD), ligado ao diretório principal. O diretório principal é indexado pelo nome do usuário e possui uma entrada que aponta para cada diretório pessoal. Esta estrutura está ilustrada na fig. 2.3.

FIGURA 2.3 - ESTRUTURA DE DIRETÓRIOS COM DOIS NÍVEIS

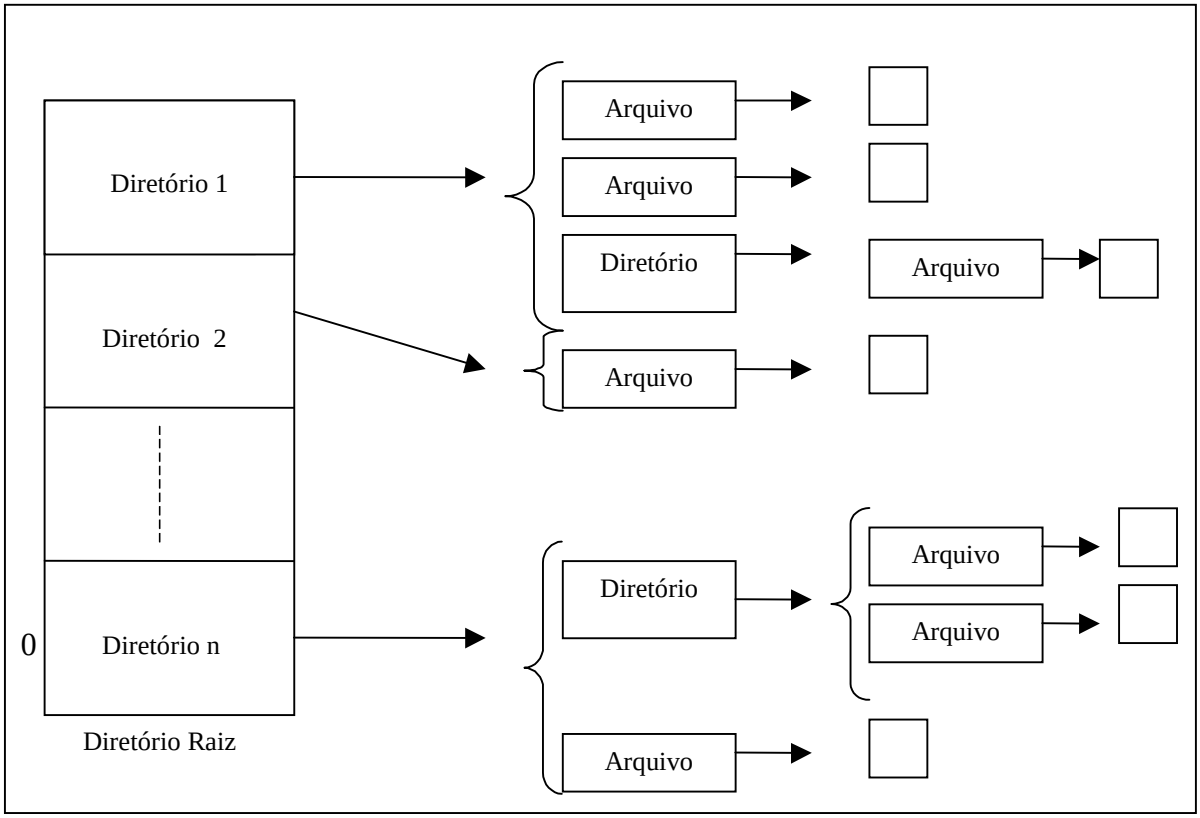


FONTE: [MAC97]

2.3.2.3 ESTRUTURA DE DIRETÓRIOS DE MÚLTIPLOS NÍVEIS

Mesmo com a estrutura de dois níveis, a organização dos arquivos em um único diretório ainda não é considerada adequada. Existe um terceiro tipo de estrutura, a de múltiplos níveis, também chamada de estrutura de diretórios em árvore, que permite ao usuário criar quantos diretórios desejar, permitindo que os arquivos fiquem melhor organizados logicamente, conforme visto na fig. 2.4. Este tipo de estrutura é o mais utilizado pela maioria dos sistemas operacionais [MAC97].

FIGURA 2.4 - ESTRUTURA DE DIRETÓRIOS EM ÁRVORE



FONTE: [MAC97]

2.3.3 NOMES DE ARQUIVOS

Nos sistemas de arquivos que utilizam estrutura de árvore de diretórios, existem regras para a formação dos nomes dos arquivos. Dois métodos são utilizados para a formação de nomes de arquivos: nome de caminho absoluto e nome de caminho relativo.

No nome de caminho absoluto, o nome do arquivo é composto pelo caminho do diretório raiz até o arquivo. Por exemplo, **/dir1/dir2/x** significa que o diretório raiz contém um diretório chamado dir1, e que este contém um subdiretório chamado dir2, e que este por sua vez contém um arquivo chamado x.

No nome de caminho relativo, utiliza-se do conceito de diretório corrente ou diretório de trabalho. Neste método, o usuário designa um diretório para ser o diretório corrente, e todos os arquivos referenciados sem o diretório raiz no caminho, são considerados relativos ao diretório corrente. Por exemplo, se o diretório corrente é /dir1/dir2, então o arquivo cujo nome absoluto é /dir1/dir2/x pode ser referenciado apenas por x [TAN87].

3 SISTEMA DE ARQUIVOS DISTRIBUÍDOS

Assim como nos sistemas centralizados (monotarefa e multitarefa), o sistema de arquivos é parte fundamental e mais visível de um sistema operacional distribuído, pois freqüentemente os usuários ou aplicações precisam manipular arquivos e estas operações devem ser de maneira uniforme, independente do tipo de dispositivo onde os arquivos estão armazenados.

Um sistema de arquivos distribuído é um sistema de arquivos onde servidores, clientes e meios de armazenamento estão dispersos por máquinas de um sistema distribuído. Nestes ambientes, considerações sobre concorrência, segurança e métodos de acesso aos recursos devem ser observadas na implementação do sistema de arquivos, para tratar questões como compartilhamento e transparência.

3.1 ESTRUTURA DOS SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

Para a compreensão da estrutura dos sistemas de arquivos distribuídos e seu funcionamento, torna-se necessário definir os termos **servidor de arquivos**, **serviço de arquivos** e **cliente**.

O **serviço de arquivos** é a especificação daquilo que o sistema de arquivos oferece como possíveis operações sobre os arquivos. Ele descreve as primitivas disponíveis, quais os parâmetros dessas primitivas e quais as tarefas realizadas por elas. O serviço de arquivos representa quais os serviços com os quais o usuário poderá contar.

Um **servidor de arquivos** é um processo que executa em alguma máquina do sistema, que auxilia a implementação do serviço de arquivos. Um sistema de arquivos distribuídos pode ter mais de um servidor de arquivos, podendo cada um oferecer um serviço de arquivos diferente. Por exemplo, um servidor poderia oferecer o serviço de arquivos do Unix e outro servidor poderia oferecer o serviço de arquivos do Windows, atendendo de forma mais abrangente diversos usuários.

Para os usuários, a maneira como está implementado o serviço de arquivos ou o fato de ele ser distribuído não é importante. O que ele tem a fazer é solicitar uma tarefa através de procedimentos especificados pelo serviço de arquivos e a tarefa será executada.

Um **cliente** é um processo que pode solicitar um serviço oferecido pelo servidor de arquivos, utilizando as primitivas implementadas por ele, disponibilizadas através de sua interface [TAN95] [TAN87].

3.2 TIPOS DE ACESSO

Processos e aplicações que armazenam informações em arquivos normalmente desejam recuperá-las posteriormente, seja para consultas, alterações ou outras operações. Em ambientes distribuídos, a forma como os arquivos são recuperados pode ser dividida em dois tipos: modelo de acesso local e modelo de acesso remoto.

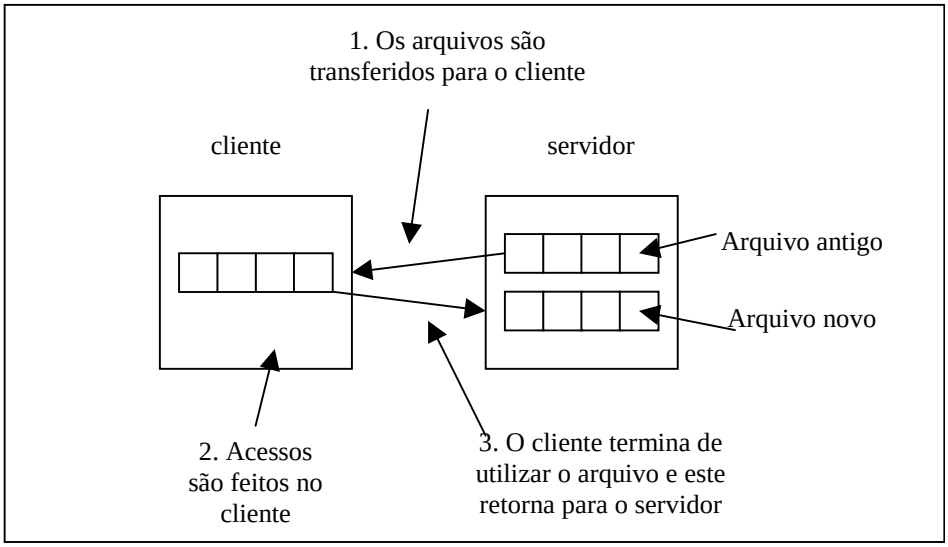
3.2.1 MODELO DE ACESSO LOCAL

O serviço de arquivos fornece duas operações: leitura e escrita de arquivos. A leitura transfere um arquivo inteiro de um dos servidores de arquivos para o cliente que o requisitou. A escrita transfere um arquivo do cliente para o servidor. A movimentação é de arquivos inteiros, em ambas as direções, e os arquivos podem ser armazenados em disco ou na memória, de acordo com a necessidade.

O modelo de acesso local é bastante simples: as aplicações buscam os arquivos necessários e utilizam-no localmente. Arquivos que são modificados ou criados pela aplicação devem ser escritos de volta ao servidor, conforme fig. 3.1.

A transferência de arquivos inteiros pode ser eficiente, mas tem a desvantagem de precisar de espaço de memória no cliente, para armazenar todos os arquivos necessários. Este aspecto pode tornar inconveniente a transferência de um arquivo inteiro quando o cliente precisar somente de uma parte dele.

FIGURA 3.1 - MODELO DE ACESSO LOCAL A ARQUIVOS



FONTE: [TAN95]

3.2.2 MODELO DE ACESSO REMOTO

Neste modelo, o serviço de arquivos fornece um grande número de operações possíveis de serem realizadas sobre os arquivos, como: leitura e escrita de parte dos arquivos, movimentação de informações dentro do próprio arquivo, verificação dos atributos dos arquivos, operações para abrir e fechar arquivos, etc. No modelo de acesso remoto, o arquivo permanece no servidor, não sendo necessária sua transferência para o cliente. Os clientes requisitam operações a serem realizadas nos arquivos, e estas operações são executadas diretamente no servidor, que é onde fica o sistema de arquivos [TAN95].

3.3 TRANSPARÊNCIA

Em sistemas de arquivos distribuídos a formação de nomes de arquivos envolve o fato de que o arquivo pode estar localizado em qualquer máquina da rede. Para que um usuário de sistema distribuído possa agir em relação aos arquivos como se estivesse trabalhando em um único computador, é necessário levar em consideração aspectos de transparência. Há duas formas relevantes de transparência: transparência quanto a localização e independência quanto a localização.

3.3.1 TRANSPARÊNCIA QUANTO À LOCALIZAÇÃO

Na transparência quanto à localização, o caminho do arquivo ou objeto não deve dar nenhuma indicação de onde ele está localizado fisicamente. Um caminho como /servidor1/dir1/dir2/x informa que o arquivo x está localizado no servidor1, mas não informa em que ponto da rede o servidor1 está localizado. Assim, o servidor1 pode ser colocado em qualquer ponto da rede sem que o caminho seja alterado.

3.3.2 INDEPENDÊNCIA QUANTO À LOCALIZAÇÃO

Imaginando que o espaço no servidor1 esteja escasso, o sistema poderia automaticamente transferir o arquivo x para o servidor2. Mas nos casos em que o primeiro componente do nome do caminho é o nome do servidor, o sistema não pode mover arquivos de um servidor para outro automaticamente. Nesta movimentação, o nome do arquivo mudaria de /servidor1/dir1/dir2/x para /servidor2/dir1/dir2/x. Todos os processos que utilizem o nome /servidor1/dir1/dir2/x param de funcionar se o caminho mudar.

Um sistema em que os arquivos podem ser movidos sem que o nome do arquivo (caminho) mude são considerados independentes quanto à localização.

Os sistemas em que o nome do servidor é embutido no caminho não são independentes quanto à localização. E os sistemas baseados em montagem remota também não são, pois não é possível mover um arquivo de um grupo (unidade de montagem) para outro, e ainda ser capaz de utilizar o nome antigo.

3.4 IDENTIFICAÇÃO EM DOIS NÍVEIS

A maioria dos sistemas distribuídos utilizam alguma forma de identificação em dois níveis. Ao ser criado um arquivo, o usuário dá a ele um nome simbólico (normalmente um conjunto de caracteres), que será utilizado para qualquer acesso posterior ao arquivo. O sistema pode gerar um nome binário para o mesmo arquivo, de modo que, ao ser referenciado o nome simbólico, o sistema procura internamente pelo nome binário atribuído ao arquivo. O papel do diretório é fornecer uma ligação entre o nome simbólico e o utilizado pelo sistema.

O nome binário dos arquivos é também conhecido como nó-i. O nó-i contém informações de onde localizar o arquivo e quais os blocos de memória utilizados por seus dados.

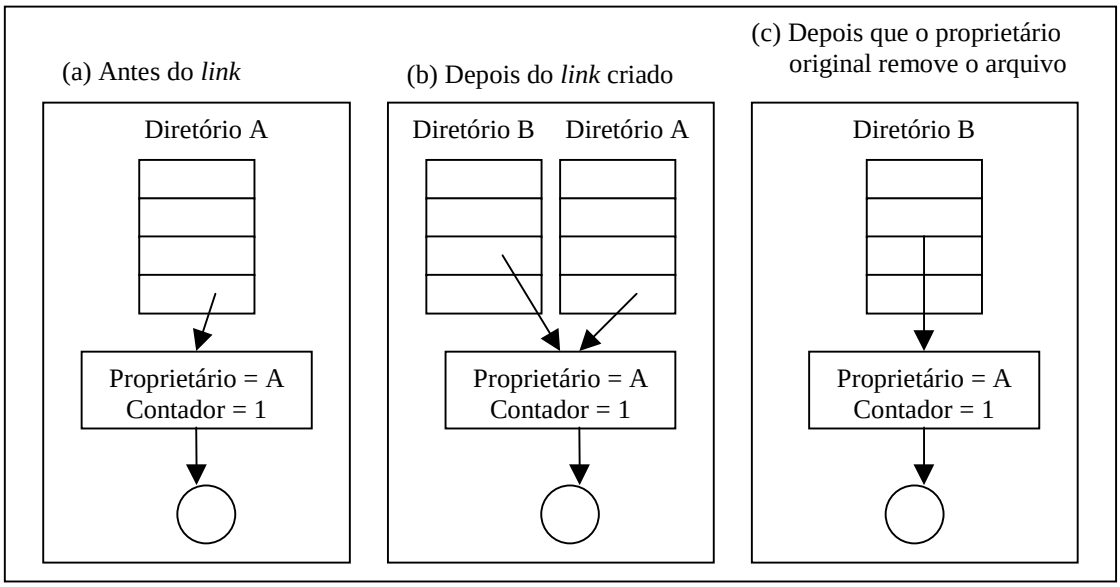
Os nomes binários podem variar de sistema para sistema. Nos sistemas compostos por diversos servidores de arquivos, o nome binário pode simplesmente ser um número de nó-i local (neste caso não é possível referenciar nenhum arquivo ou diretório de outros servidores).

Uma outra forma de identificar o arquivo é ter no nome binário a especificação tanto do servidor quanto de um arquivo específico neste servidor. Esta identificação permite que um servidor possa referenciar-se a arquivos de outro servidor, fornecendo um mecanismo para compartilhamento de arquivos. Isto pode ser feito através de uma ligação simbólica (*links*).

3.5 COMPARTILHAMENTO DE ARQUIVOS

Quando diversos usuários precisam utilizar o mesmo arquivo, este arquivo precisa ser compartilhado. Uma forma de compartilhamento é a criação de *links*, onde um arquivo aparecerá simultaneamente em diretórios diferentes que pertencem a diferentes usuários. Imagine um arquivo x, pertencente a um diretório "A", e compartilhado com um diretório "B" (fig. 3.2). Neste caso, o arquivo x no diretório "B" seria uma **ligação** (*link*).

FIGURA 3.2 - SITUAÇÃO DE UM DIRETÓRIO COM UM *LINK* PARA UM ARQUIVO DE OUTRO DIRETÓRIO



FONTE: [TAN95]

Um arquivo pode possuir múltiplos links. Assim, diferentes nomes de arquivos podem ser acessados por diversos usuários no acesso às informações de um único arquivo [TAN95] e [MAC97].

O compartilhamento de arquivos pode trazer alguns problemas. Se os diretórios armazenam em suas entradas realmente endereços físicos dos blocos de um arquivo, ao ser feita uma ligação, como no exemplo da fig. 3.2, uma cópia desses endereços é feita no diretório "B", no momento da criação do *link*. Se "A" ou "B" acrescentarem algo ao arquivo, os novos blocos serão acrescentados apenas na entrada do diretório que modificou o arquivo.

Uma maneira de resolver isso é fazer com que os endereços dos blocos físicos não fiquem na entrada dos diretórios, mas em uma estrutura de dados associada ao arquivo. Neste caso, os diretórios devem apontar para esta estrutura. O *Unix* implementa este tipo de compartilhamento e utiliza como estrutura um nó-i.

Outra solução é fazer com que seja criado um novo arquivo no diretório do usuário que deseja acessar o arquivo de outro usuário. Dessa forma, o novo arquivo conterá o nome do caminho do arquivo com o qual ele é ligado. O novo arquivo criado é marcado como sendo um *link*, de forma que quando o usuário fizer uma operação de leitura do arquivo, o sistema busca o nome do arquivo para fazer a leitura. Este método é denominado **ligação simbólica**.

Estes métodos também tem seus problemas. No primeiro deles, um arquivo compartilhado tem um campo onde é armazenado o proprietário do arquivo. A ligação não muda o proprietário, apenas muda o contador de ligação, para saber quantas entradas de diretório apontam para o arquivo.

Tomando o exemplo da fig. 3.2, se "A" apagar o arquivo e a estrutura correspondente aos blocos desse arquivo também for removida, o diretório "B" terá uma entrada apontando para uma estrutura inválida. A melhor solução é fazer com que o sistema detecte que o arquivo ainda está em uso através do contador e, ao remover a entrada do diretório "A", mantenha a estrutura de blocos do arquivo intacta, com o contador em 1, conforme fig. 3.2. Neste caso, o diretório "B" é o único a ter uma entrada para um arquivo cujo proprietário é "A". Quando "B" remover este arquivo e o contador chegar em zero, a estrutura de dados que contém os blocos do arquivo é então removida.

Com ligações simbólicas, somente o verdadeiro proprietário tem um ponteiro para a estrutura de dados dos blocos do arquivo. Os usuários que foram ligados ao arquivo tem apenas nomes de caminho. Qualquer tentativa de usar este arquivo por uma ligação simbólica falhará, pois o arquivo não poderá ser encontrado.

As ligações simbólicas enfrentam um problema de *overhead*. Para ser encontrada a estrutura de dados dos blocos do arquivo, o caminho de uma ligação precisa ser analisado, componente a componente. Esta busca causa diversos acessos ao disco. A grande vantagem desse tipo de ligação é que podem ser usadas para ligar arquivos de máquinas que estejam em qualquer lugar, através de um endereço da rede de comunicação de onde está o arquivo e o caminho de localização desse arquivo nesta máquina.

O compartilhamento de arquivos em ambientes distribuídos pode causar alguns problemas pelo fato de diversos usuários poderem realizar operações no mesmo arquivo e porque pode haver concorrência nestas operações. Por isso, quando dois ou mais usuários compartilham o mesmo arquivo, é necessário definir semântica de leitura e escrita, para não ocorrer de um usuário obter o conteúdo antigo de um arquivo que foi alterado.

3.5.1 SEMÂNTICA UNIX

Em sistemas com um único processador que permitem compartilhamento de arquivos, a semântica normalmente estabelece que uma operação READ sempre obterá o valor do último WRITE executado no arquivo. Dessa forma, sempre é retornado o valor mais recente. Para que isso funcione, o sistema mantém uma ordenação absoluta de todas as operações [TAN95].

Em sistemas distribuídos, esta semântica pode ser facilmente adquirida caso o sistema possua um único servidor de arquivos, e os clientes não coloquem arquivos na *cache*. Todas as operações de leitura e escrita vão diretamente para o servidor de arquivos e são processadas em ordem sequencial. Podem ocorrer problemas quando acontecer um retardo na rede, e por exemplo, um *read* que foi executado um microssegundo após um *write* chegue antes no servidor, obtendo o valor antigo.

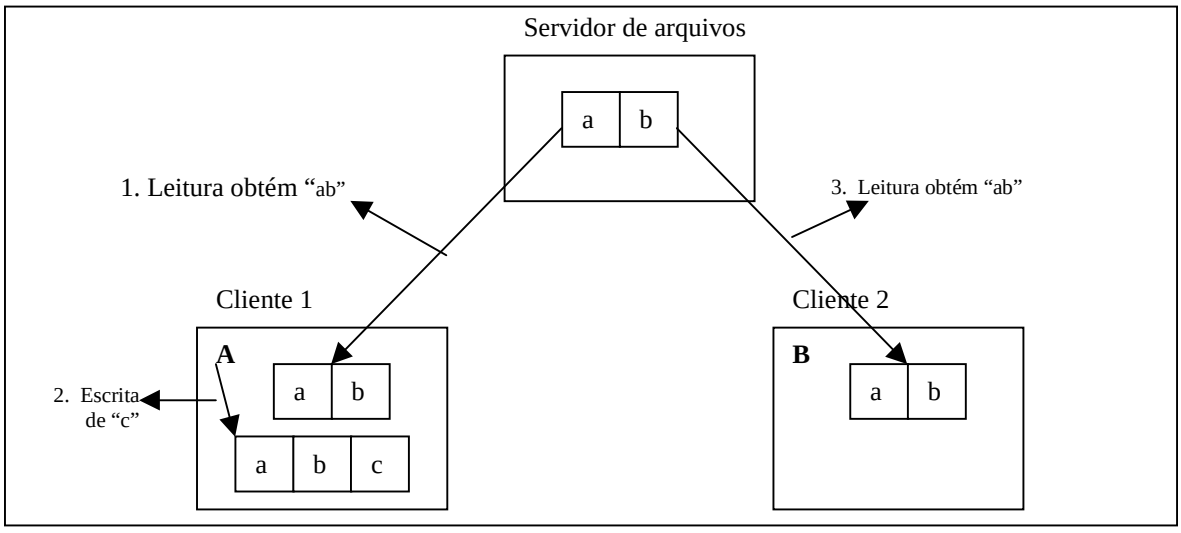
Manter todas as informações no disco do servidor, e fazer com que todas as requisições passem pelo servidor para serem executadas pode resultar em problemas de

performance. Isto ocorre porque, antes que um cliente possa ler um arquivo, ele precisa ser transferido do disco do servidor para a sua memória principal, e daí para a memória principal do cliente, através da rede. Ambas as transferências consomem tempo. Este problema pode ser amenizado armazenando (*caching*) na memória principal do servidor os arquivos mais recentemente utilizados, formando a chamada *cache* do servidor. Dessa forma, um arquivo requisitado pelo cliente, lido diretamente da memória do servidor, elimina a necessidade de transferência do disco, embora ainda tenha que transferir o arquivo através da rede.

Uma forma para eliminar a transferência através da rede é manter uma *cache* na memória do cliente também. Utilizar a memória principal do cliente ou o espaço de seu disco é a escolha entre espaço e performance. O disco tem mais capacidade, porém seu acesso é mais lento. A memória possibilita acesso mais rápido, porém é mais limitada quanto a capacidade de armazenamento.

Voltando a questão de compartilhamento, se um cliente mantém localmente um arquivo em sua *cache* e o modifica, outro cliente que leia este arquivo logo em seguida do servidor obterá um arquivo obsoleto, conforme visto na fig. 2.7.

FIGURA 3.3 - ARQUIVO MANTIDO EM *CACHE* DE UM CLIENTE E LIDO POR OUTRO CLIENTE



FONTE: [TAN95]

Uma alternativa para solucionar este problema é enviando ao servidor imediatamente todas as alterações feitas nos arquivos armazenados em *cache*, mas este método é ineficiente.

Para esta alternativa, existe o algoritmo ***write through***. Ele define que, ao ser alterado um arquivo, ele permanece na *cache*, mas também é enviado imediatamente ao servidor. Isso pode causar alguns problemas. Se um processo de um cliente modificar um arquivo, ele o mantém na *cache* e envia uma cópia ao servidor. Se um processo de outro cliente abrir o arquivo e fizer modificações, enviando também uma cópia ao servidor e o cliente anterior abrir novamente o arquivo que está em sua *cache*, ele terá um valor obsoleto. Isso pode ser resolvido, fazendo com que o gerenciador de *cache* verifique a situação do arquivo junto ao servidor. Poderia ser comparado o instante da última modificação da cópia da *cache* com a cópia do servidor, ou um número de versão. Se o número ou o instante forem os mesmos, a *cache* está atualizada. Caso contrário, o gerenciador busca o arquivo no servidor. Nesta comparação, a quantidade de dados transmitidos pela rede é pequena, porém usa algum tempo. O algoritmo *write through* auxilia a leitura, mas gera um grande tráfego nas escritas.

Uma outra alternativa é o algoritmo de **escrita retardada**. Este algoritmo sugere que, quando um cliente fizer escrita em um arquivo, ele simplesmente envia ao servidor uma nota indicando que o arquivo foi atualizado e só envia o arquivo de volta ao servidor dentro de determinados períodos de tempo.

3.5.2 SEMÂNTICA DE SESSÃO

Uma outra alternativa é conhecida como **semântica de sessão**, e define que as alterações feitas em arquivos são visíveis apenas para os processos que modificam os arquivos. Somente após o arquivo ser fechado é que estas alterações são enviadas ao servidor. Isto não muda o que acontece na fig. 3.3, mas estipula um modo de comportamento. Um cliente obtém o valor original de um arquivo do servidor como sendo o correto. Quando o cliente que fez as modificações fechar o arquivo, uma cópia deste é enviada ao servidor, de forma que as leitura subseqüentes feitas ao arquivo obterão o conteúdo atualizado. No uso desta semântica, se dois clientes modificarem simultaneamente um arquivo, o resultado final deste depende de quem fechou por último o arquivo. O primeiro cliente faz as alterações, fecha o arquivo e envia uma cópia para o servidor. O segundo cliente faz o mesmo. O que fechar por último o arquivo teve suas modificações mantidas. Logo, a semântica de sessão, com este algoritmo, que é denominado ***write-on-close***, não garante que todas as leituras retornarão o valor mais recente do arquivo.

Para melhorar a consistência utilizando esta semântica, pode-se utilizar um algoritmo denominado de **controle centralizado**. Ele define que, quando um arquivo é aberto, a máquina que o acessou faz uma notificação ao servidor, através de uma mensagem. O servidor mantém um controle sobre quem tem arquivos abertos e se estes arquivos são somente para leitura, escrita ou leitura/escrita.

Se for somente para leitura, não há problemas em outra máquina acessar o arquivo para leitura, porém, não deve ser aberto para escrita. Da mesma forma, quando um cliente abrir um arquivo para escrita, outros acessos a este arquivo devem ser proibidos. Quando um arquivo é modificado, no momento em que ele é fechado e enviado ao servidor, este atualiza suas tabelas de acesso aos arquivos.

3.5.3 SEMÂNTICA DE ARQUIVOS IMUTÁVEIS

Uma outra alternativa para a semântica de compartilhamento de arquivos é a de **arquivos imutáveis**. O uso desta semântica faz com que os arquivos sejam imutáveis, não permitindo que um arquivo seja aberto para escrita. As únicas operações realizadas sobre os arquivos são leitura (*read*) e criação (*create*).

Neste método, a criação de um arquivo em determinado diretório, com o mesmo nome de um arquivo já existente, deixa o arquivo antigo inacessível (pelo menos sob este nome). Isso elimina o problema de um arquivo estar sendo escrito por um processo e lido por outro processo simultaneamente, mas permanece o problema de dois processos tentarem substituir o mesmo arquivo ao mesmo tempo. Umas das maneiras de lidar com isso, como na semântica de sessão, é definir que o processo que terminar por último a geração do arquivo substitua o arquivo antigo.

Outro problema com a semântica de arquivos imutáveis é que um arquivo pode estar sendo substituído enquanto outro processo faz a leitura deste arquivo. Neste caso, pode-se utilizar meios que garantam a continuação da leitura do arquivo antigo, mesmo que ele não exista mais em nenhum diretório e a substituição do mesmo arquivo por um novo. Ou pode-se fazer com que seja detectada uma modificação no arquivo e gerar uma falha.

3.5.4 SEMÂNTICA DE TRANSAÇÕES

Outro método de semântica de compartilhamento de arquivos é a **semântica de transações**. Essa semântica define que um processo que deseja acessar um arquivo ou grupo de arquivos deve executar uma chamada de “início de transação”. A partir do início da transação, toda a leitura ou escrita definida dentro da transação é executada sequencialmente, sem interferência de nenhum outro processo ou transação concorrente. Ao terminar o trabalho, o processo envia um “fim de transação”. Se duas ou mais transações iniciarem ao mesmo tempo, o sistema deve garantir que o resultado final é o mesmo que o obtido se as duas transações fossem executadas em alguma ordem sequencial indefinida. Por exemplo, se em uma operação bancária de depósito em conta corrente, dois processos tentarem adicionar simultaneamente uma determinada quantia, as operações devem ser agrupadas em uma única transação, fazendo com que o resultado final obtenha o valor correto. Se a conta continha 100 dólares e um dos processos quer depositar mais 50 dólares e o outro processo deseja depositar 20 dólares, as operações devem ser executadas de modo que o resultado não seja nem 150 dólares e nem 120 dólares (advindos de uma leitura no valor antigo de 100 dólares), mas sim 170 dólares, realizando a leitura dos 100 dólares, a escrita de mais 20 (ou 50) dólares, a nova leitura do valor atualizado e mais uma escrita de 50 (ou 20) dólares, resultando no montante correto das duas transações [TAN95].

3.6 REPLICAÇÃO DE ARQUIVOS

Alguns sistemas de arquivos distribuídos oferecem o serviço de replicação de arquivo. Isso significa ter várias cópias de arquivos selecionados, que serão mantidos em servidores distintos.

A replicação de arquivos traz alguns benefícios:

a) torna o sistema mais confiável, pelos diversos *backups* que possui. Se um dos servidores tiver problemas, o arquivo fica acessível através de outro servidor. Ou se ocorrer uma falha irreversível no sistema de arquivos de um servidor, não haverão perdas de dados;
b) é possível acessar um arquivo mesmo se um servidor tiver problemas;

c) a replicação pode trazer também ganhos de performance. Se um servidor estiver muito carregado, pode-se acessar um arquivo através de outro servidor que esteja mais liberado.

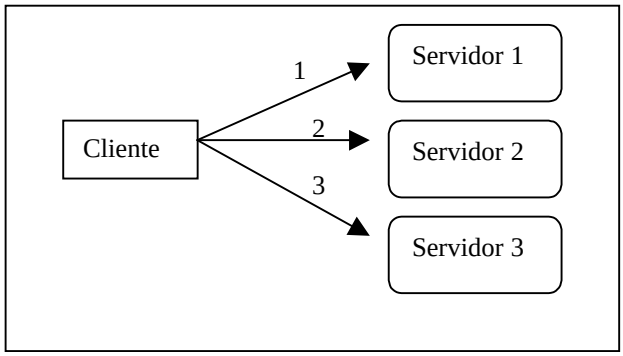
3.6.1 FORMAS DE IMPLEMENTAR A CRIAÇÃO DE ARQUIVOS REPLICADOS

Se a replicação ficar sob a total responsabilidade do sistema, controlando tudo que for necessário para que ela ocorra, então diz-se que a replicação é transparente. Existe também a alternativa de os clientes ficarem a par do processo de replicação, podendo inclusive controlá-lo. Neste caso a replicação não é transparente.

3.6.1.1 REPLICAÇÃO EXPLÍCITA

Uma das maneiras de implementar a replicação é fazer com que o programador controle todo o procedimento. Quando um processo cria um arquivo, este é criado em um servidor específico. A partir daí, ele pode, se necessário, fazer cópias adicionais do arquivo em outros servidores. Se o servidor de diretórios permitir múltiplas cópias de um arquivo, o endereço de cada cópia na rede pode ser associada ao nome (fig. 3.4), de forma que ao ser buscado um nome de arquivo , todas as cópias poderão ser encontradas. Quando um arquivo é acessado, tenta-se sucessivamente todas as cópias, até encontrar uma disponível.

FIGURA 3.4 ENDEREÇO DE ARQUIVOS REPLICADOS COM ASSOCIAÇÃO AO NOME



Arquivo	1.14	2.16	3.19
Prog.c	1.21	2.43	3.41

Nome simbólico

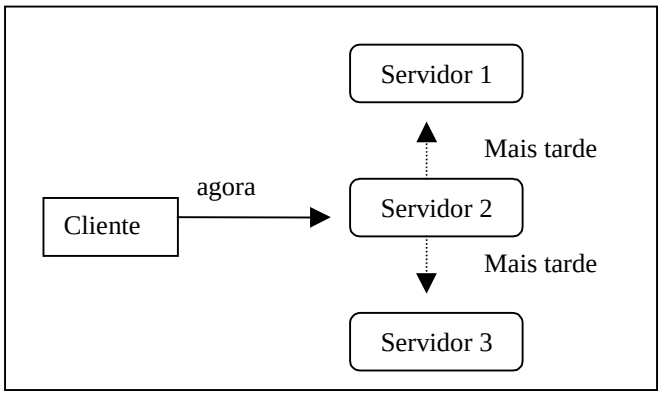
Endereços binários múltiplos (para Servidor1, servidor2 e servidor3

FONTE: [TAN95]

3.6.1.2 REPLICAÇÃO RETARDADA

Na replicação retardada, é feita apenas uma cópia de cada arquivo em algum servidor. A replicação ocorre em *background*. Quando o servidor tiver um tempo livre, ele se encarrega de fazer outras cópias para outros servidores, conforme mostrado na fig. 3.5. O sistema deve ter mecanismos para recuperar cada uma destas cópias, quando necessário. Deve-se levar em consideração o fato de um arquivo poder ser modificado antes das cópias terem sido feitas.

FIGURA 3.5 - REPLICAÇÃO RETARDADA DE UM ARQUIVO

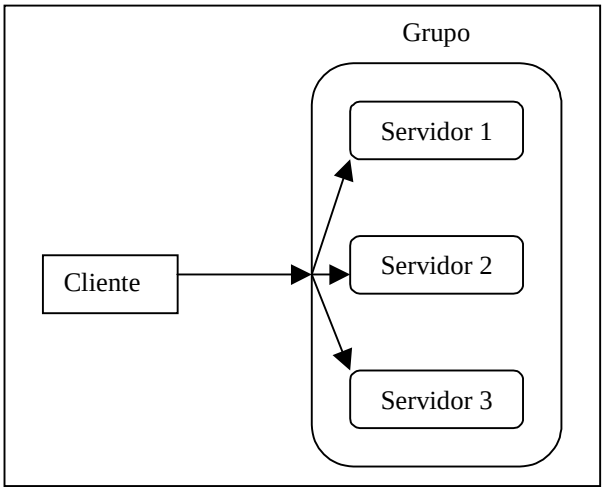


FONTE: [TAN95]

3.6.1.3 REPLICAÇÃO UTILIZANDO COMUNICAÇÃO EM GRUPO

Neste método, a replicação ocorre no momento que o arquivo original é criado, conforme fig. 3.6. Ao ser utilizada uma chamada *write*, esta é transmitida para todos os servidores, onde são feitas cópias extras do arquivo. Neste caso, é endereçado um grupo de servidores e não apenas um servidor como na replicação retardada.

FIGURA 3.6 - USO DE UM GRUPO NA REPLICAÇÃO DE UM ARQUIVO



FONTE: [TAN95]

3.6.2 ATUALIZAÇÃO DE ARQUIVOS REPLICADOS

Quando o sistema utiliza o conceito de replicação de arquivos deve-se levar em conta a atualização das cópias em caso de modificação nos arquivos. Enviar simplesmente uma mensagem de atualização a cada cópia pode não ser muito eficiente, pois se o processo responsável por essa tarefa falhar, alguns arquivos podem não ser modificados, resultando na possível leitura de conteúdos obsoletos em algumas operações.

Dois algoritmos bastante conhecidos para resolver este problema estão descritos a seguir.

3.6.2.1 REPLICAÇÃO DA CÓPIA PRINCIPAL

Neste método, um dos servidores é designado a ser o servidor principal, enquanto os outros são considerados secundários. Qualquer alteração em um arquivo replicado é feita no servidor principal. Este se encarrega de enviar comandos aos servidores secundários, indicando que devem fazer as alterações necessárias.

Para resolver o problema de falhas no servidor principal, cada alteração é gravada também em memória estável, antes da modificação no servidor principal. Quando o servidor se recuperar da falha, ele verifica se alguma modificação estava em andamento quando ela ocorreu. Em caso positivo, o procedimento pode ser retomado. Dessa forma, em algum momento todos os servidores receberão a atualização do arquivo. O problema com este método é que nenhuma alteração em arquivos replicados é atualizada quando o servidor principal estiver parado por algum problema.

3.6.2.2 REPLICAÇÃO SELETIVA

O propósito deste método é fazer com que um cliente que deseja ler ou escrever em um arquivo replicado, deva, primeiramente, solicitar e adquirir permissão de vários servidores para depois poder realizar a operação. O número de servidores a serem contatados pode variar. Um bom exemplo é fazer com que metade dos servidores mais um precise dar o consentimento ao cliente.

Quando um arquivo replicado é criado, ele recebe um número de versão, que é igual para todas as suas cópias. Para atualizar o arquivo, o cliente então requisita permissão à

maioria dos servidores. Se for concedido o direito, o arquivo é atualizado e uma nova versão é gerada em todas as cópias atualizadas.

Para a leitura de um arquivo, um cliente faz novamente contato com a maioria dos servidores solicitando que eles enviem o número da versão. Se metade mais um dos arquivos tiverem a mesma versão, significa que estão atualizados, pois para terem sido alterados na versão anterior, também houve essa mesma verificação na maioria dos servidores.

4 ESTUDOS DE CASO

Para ilustrar alguns dos conceitos e demonstrar a utilização dos métodos apresentados e estudados anteriormente, serão descritos alguns dos sistemas de arquivos distribuídos encontrados na literatura.

4.1 SISTEMA DE ARQUIVOS ANDREW

O sistema Andrew (AFS) foi desenvolvido na Carnegie University, e seu nome se deve aos fundadores da Universidade, Andrew Carnegie e Andrew Mellon.

O AFS tem a característica de tratar de 5.000 a 10.000 estações de trabalho, sendo que um grande quantidade delas podem estar ativas ao mesmo tempo.

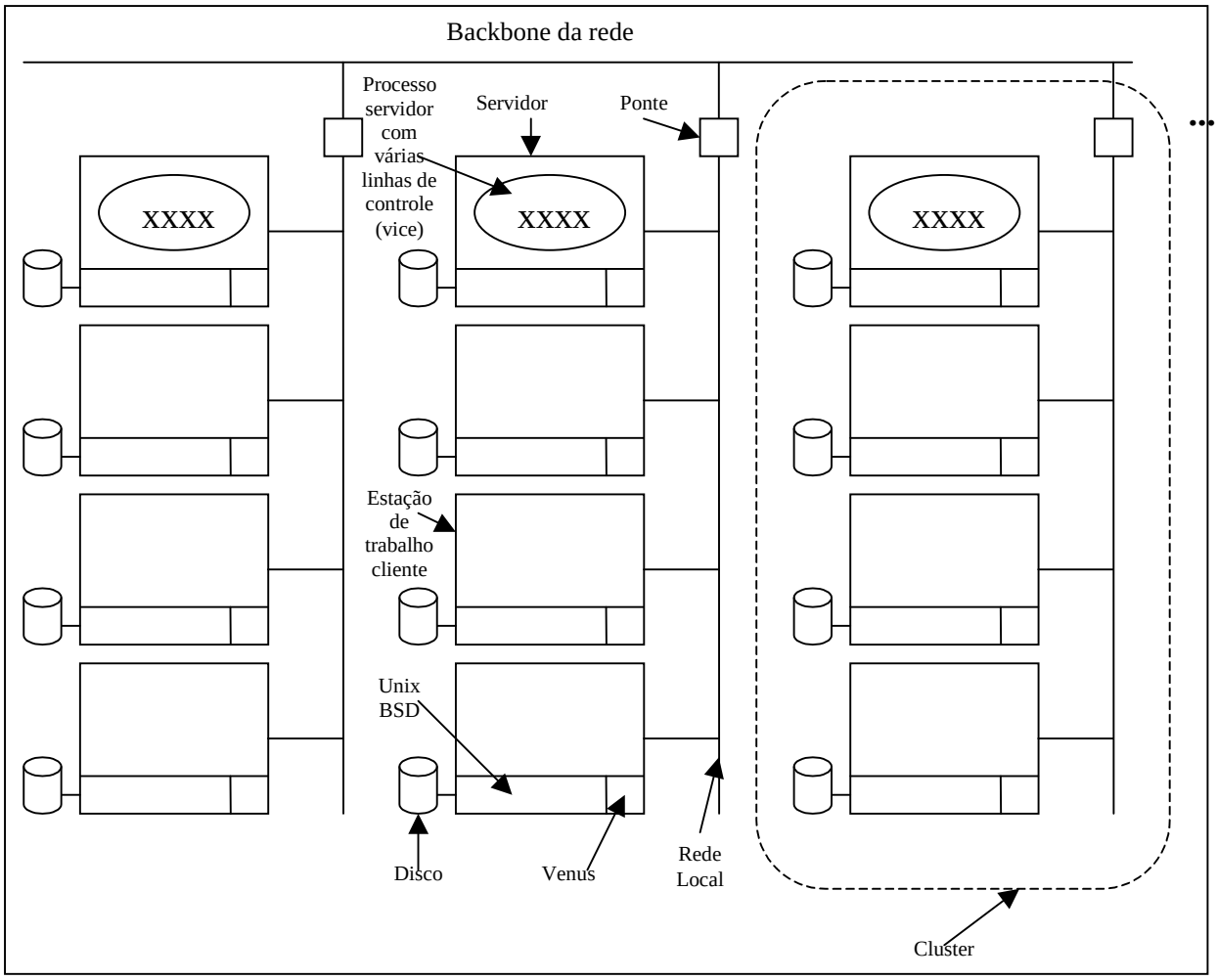
Baseando-se em [TAN95], [VAH96] e [SIL94] serão descritos o funcionamento e estrutura do AFS.

4.1.1 ARQUITETURA DO AFS

O AFS é constituído por *clusters*. Cada *cluster* contém um servidor de arquivos e diversas estações clientes. Conforme mostra a fig 4.1, procurou-se fazer com que o tráfego fosse local a um único *cluster*, reduzindo a carga de transferência na rede.

Máquinas clientes e servidoras não se distinguem fisicamente, e rodam versões do sistema operacional *Unix* de *Berkeley*. Acima do *kernel*, os clientes e servidores rodam softwares diferentes. Os servidores executam um programa chamado *Vice*, que trata as chamadas de operações sobre os arquivos, vindas dos clientes. Os clientes executam editores de textos, gerenciadores de janelas e outros softwares do *Unix*. Existe nos clientes, também, um programa chamado *Venus*, que funciona como gerenciador de *cache* e faz a interface entre o cliente e o *Vice*.

FIGURA 4.1 - ARQUITETURA DO AFS



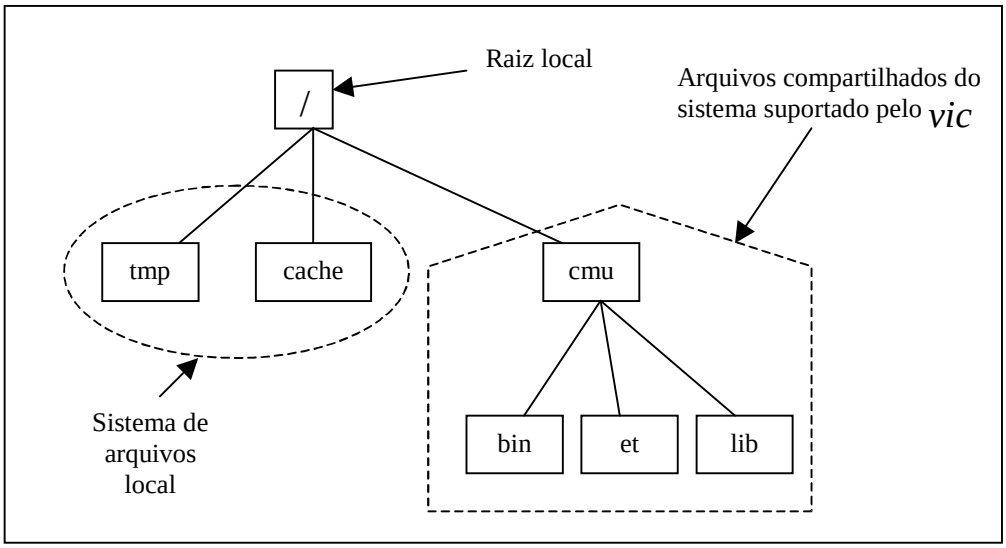
FONTE: [SIL94]

A estrutura de diretório do AFS é uma estrutura de árvore, que contém um diretório /*cmu*, conforme visto na fig. 4.2. Este diretório é idêntico em todas as estações de trabalho e o seu conteúdo é suportado pelo AFS via servidores *vice*. Outros diretórios e arquivos são apenas locais, contém arquivos temporários, *cache*, arquivos de inicialização da estação, etc., e não são compartilhados. Os arquivos e diretórios a serem compartilhados são implementados através de ligações simbólicas (*links*), a partir do diretório /*cmu*.

Todas as estações de trabalho dos clientes são equipadas com um disco rígido e o AFS procura fazer com que as estações sejam o mais independente possível, interagindo o mínimo com o restante do sistema. Um arquivo aberto é transportado para a *cache* do cliente que solicitou o arquivo, é inserido em um diretório local /*cache* e tratado como arquivo local pelo sistema operacional. O código dos comandos *read* e *write* do *Unix* não foram alterados,

apenas modificados para manipular a interação entre clientes, *cache* e servidor de arquivos, de modo que os arquivos são utilizados sem levar em consideração o restante do sistema.

FIGURA 4.2 - ESTRUTURA DE DIRETÓRIOS DO AFS



FONTE: [TAN95]

Por tratar um grande número de usuários, a questão de segurança é fundamental para o AFS, já que os usuários podem dar *boot* a partir de qualquer estação. Toda comunicação feita entre servidores e estações é criptografada e suportada por hardware.

Os diretórios são protegidos por listas de controle de acesso e os arquivos possuem os nove bits *rwx* de proteção do *Unix*, para o dono do arquivo, grupo de usuários e para os demais usuários.

Pelo fato de as estações de trabalhos possuírem apenas arquivos temporários na *cache* de arquivos, somente os servidores precisam ser mantidos e ter seus *backups* em dia.

4.1.2 A SEMÂNTICA DO AFS

Além de arquivos e diretórios, o AFS suporta o conceito de volume e célula. Volume são os diretórios gerenciados em conjunto. Normalmente, cada volume é uma coleção de diretórios pertencentes a algum usuário, outros são utilizados para guardar código executável e outras informações dos sistema, podendo os volumes serem somente de leitura ou de leitura e escrita.

A célula é uma entidade administrativa, como um departamento ou uma empresa. Pode-se dizer que a célula é um conjunto de volumes colocados juntos em alguns pontos de montagem.

A semântica utilizada pelo AFS é bem parecida com a semântica de sessão. Quando um arquivo é aberto, ele é copiado para o disco local, no diretório /*cache* da estação que o chamou. Qualquer leitura e escrita no arquivo é feita na cópia em /*cache*. Se um processo abrir um arquivo já aberto, o que ele enxerga depende do local onde se encontra.

Os processos localizados na mesma estação enxergam /*cache*, não importando o estado do arquivo. Em uma única estação, a semântica aplicada é a do *Unix*, pois os módulos do sistema se enxergam como se estivessem em um sistema *Unix* comum, com um único processador.

Os processos de outra estação enxergam o arquivo original do servidor. Somente depois do arquivo ter sido fechado e enviado de volta ao servidor é que seu conteúdo é atualizado para outras estações também.

Após um arquivo ter sido fechado ele é mantido na *cache* para o caso de ser novamente utilizado. Ao reabrir um arquivo da *cache*, torna-se necessário verificar se o conteúdo ainda é válido. Para que isso se torne possível, toda vez que *Venus* carrega um arquivo para a *cache*, ele informa a *Vice* se deve se preocupar com aberturas subsequentes, vindas de outras estações. Caso positivo, *Vice* mantém uma entrada em uma tabela, que contém a localização da *cache* do arquivo; caso outro processo de qualquer ponto do sistema tentar abrir o arquivo, *Vice* informa a *Venus* que a entrada da *cache* deve ser marcada como inválida. Se o arquivo estiver corretamente em uso, o processo que está utilizando o arquivo continua a fazê-lo. Mas se outro processo tentar abri-lo, *Venus* deve verificar se a entrada correspondente da *cache* ainda é válida, caso contrário, deve ser feita uma cópia do servidor. Se a estação tiver problemas de funcionamento e depois voltar, todas as entradas de *caches* são marcadas como inválidas por segurança.

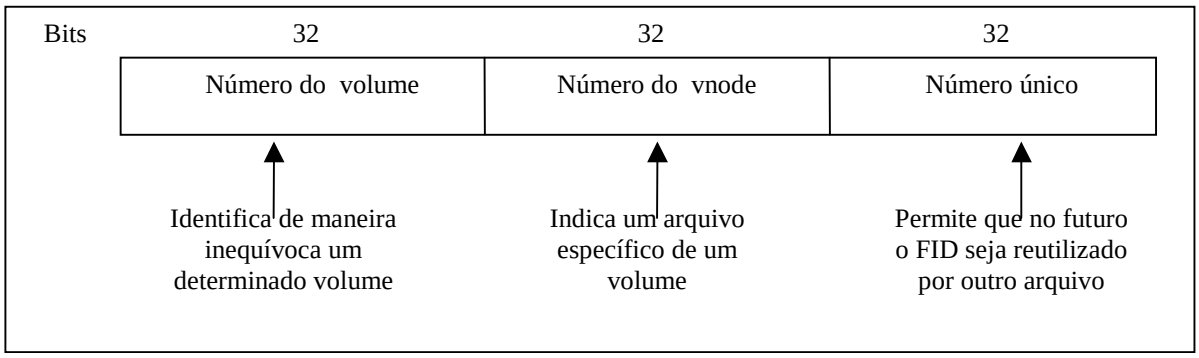
4.1.3 IMPLEMENTAÇÃO DO AFS

Todas as estações de trabalho utilizam uma cópia do *Unix* de Berkeley, muito parecida com o sistema-padrão com um único processador. A principal diferença é a adição

de *Venus* no *Kernel* e o direcionamento das chamadas *Open* e *Close* para o *Venus*, para a realização da gerência de *cache*. Nas chamadas *Open*, *Venus* retorna um descritor de arquivo apontando para um nó-i de um arquivo em /*cache*, em vez de um arquivo no servidor de arquivos.

As aplicações tratam os nomes dos arquivos da mesma forma que o padrão do Unix, mas internamente, *Venus* e *Vice* utilizam um sistema de identificação em dois níveis, onde a busca de nomes de caminho em um diretório cria estruturas chamadas **FIDs** (identificadores de arquivos) no lugar do número de nó-I, conforme mostrado na fig. 4.3.

FIGURA 4.3 - ESTRUTURA DE UM FID



FONTE: [TAN95]

Uma FID tem três campos de 32 bits cada um. O primeiro campo identifica um único volume do sistema. Este campo indica qual o volume que contém o arquivo. O segundo campo é o número do *vnode*, um índice para as tabelas do sistema relativas ao volume especificado. O terceiro campo é o número *Unique*, utilizado para reaproveitar, no caso de um arquivo ser apagado, o seu *vnode*, sob um número *Unique* diferente, de forma a detectar e rejeitar qualquer FID antigo que esteja perdido.

Todos os servidores V*ice* possuem uma base de dados com todos os volumes do sistema. Os volumes podem migrar entre os servidores, de forma que as bases de dados devem ser atualizadas de tempos em tempos. Na migração, um volume é copiado para outro servidor e depois eliminado do local de origem. Pode-se também fazer a duplicação de volumes de leitura, que ocorre da mesma forma que a migração, mas o arquivo não é eliminado do local de origem, pois é utilizado para a geração de *backups*.

O protocolo entre *Vice* e *Venus* utiliza a FID para identificar os arquivos. Quando uma FID chega no *Vice*, o número do volume é procurado na base de dados.

4.1.4 DESCRIÇÃO DO ACESSO AOS ARQUIVOS

Uma aplicação executa uma chamada *open*. Esta chamada chega até o *Venus*, que verifica o nome do caminho. Se o caminho começar com */CMU*, então o arquivo é compartilhado. Caso contrário, o arquivo é local e é tratado de maneira usual. Se o arquivo for compartilhado, é feita uma análise do seu nome, cada componente é buscado até que o FID seja encontrado. Então, através do FID, *Venus* verifica a *cache*, podendo ter como resultado:

a) o arquivo está na *cache* e é válido;
b) o arquivo está na *cache* e não é válido;
c) o arquivo não está presente na *cache*.

No primeiro caso, a cópia que se encontra na cache é utilizada. No 2º caso, *Venus* verifica com *Vice* se o arquivo foi modificado desde o momento de sua carga. Isso porque um outro processo pode ter aberto o arquivo e escrito nele, ou a estação pode ter sido reinicializada sem nenhuma modificação ter sido feita no servidor ainda. Neste caso, utiliza-se a própria cópia da *cache*. Se foi verificado modificação, uma nova cópia é transferida a partir do servidor. Nos três casos, uma cópia do arquivo estará no disco local em /*cache* e suas entradas na tabela marcadas como válidas.

Quando uma aplicação executa uma chamada *close*, *Venus* interpreta a chamada, verifica se houve alteração no arquivo e, caso positivo, faz uma cópia do arquivo de volta ao servidor. As chamadas *read* e *write* são tratadas de forma usual. *Venus* mantém também uma *cache* que mapeia nomes de caminhos e FIDs.

Então, ao analisar um nome de caminho para encontrar o FID correspondente, primeiro verifica se está na *cache*. Se estiver, a busca é cancelada e o FID da *cache* é utilizado. Se um arquivo cujo FID estiver na *cache* foi eliminado e substituído por outro arquivo, ele vai ter o número *unique* diferente do número anterior, tornando o FID inválido. Neste caso, *Venus* apaga a entrada da *cache* correspondente ao (caminho, FID) e analisa o nome desde o início.

O *Vice* roda um único programa com diversas linha de controle, em cada uma das máquinas servidoras. Para cada linha de controle, está associada uma chamada de serviço. As operações disponíveis a serem realizadas permitem movimentação de arquivos, em ambas as direções, bloqueio e desbloqueio de arquivos, gerenciamento de diretórios , etc

4.2 O AMOEBA

O Amoeba é considerado um sistema operacional distribuído por fazer um conjunto de processadores e dispositivos de entrada/saída funcionarem como se fossem um único computador.

O Amoeba surgiu em uma universidade em Amsterdã, na Holanda (*Vrije Universiteit*). Quem iniciou o projeto foi o professor Andrew S. Tanenbaum, em 1981, juntamente com 3 alunos de doutorado: Fram Kaashoek, Sape J. Mullender e Robert Van Renesse. Em 1983 estava pronta a 1ª versão do Amoeba. A partir de 1984 o projeto desmembrou-se com a criação de um 2º grupo de pesquisas, liderado por Mullender. Nos anos seguintes o estudo foi estendido a outros centros de pesquisa na Inglaterra e na Noruega. O texto abaixo descreve a versão 5.0 do Amoeba, segundo [TAN95].

4.2.1 OBJETIVOS DO AMOEBA

O objetivo principal do Amoeba é a transparência. Todos os recursos são controlados pelo sistema. Os processos individuais terão disponibilidade aos recursos por curtos períodos de tempo.

O objetivo secundário do Amoeba é disponibilizar um ambiente de testes para programação paralela e programação distribuída.

4.2.2 ARQUITETURA DO SISTEMA OPERACIONAL AMOEBA

O Amoeba foi projetado para funcionar em um ambiente de hardware onde tem-se disponível um grande número de processadores, cada um com muitos megabytes de memória. Mesmo assim, é flexível para certas adaptações de ambientes a qual se deseja testar .

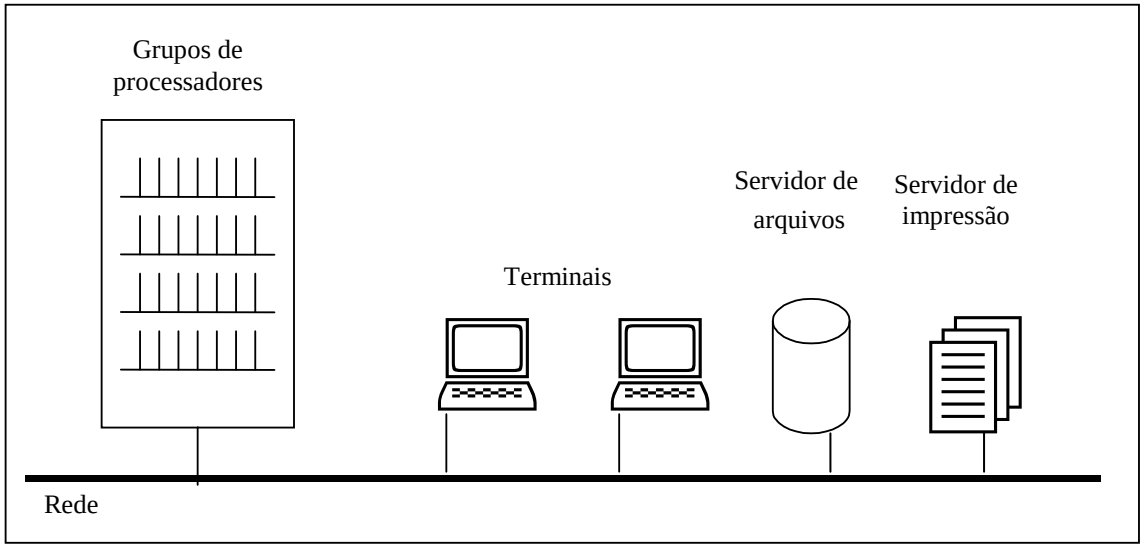
O projeto do Amoeba baseia-se em um conjunto de processadores e terminais, por onde os usuários acessam o sistema. A idéia é fazer com que, no lugar de adquirir diversas

estações de alta performance, pode-se adquirir, pelo mesmo preço, um número de processadores de alta performance, organizá-los em grupos e um número de terminais mais simples. Com essa organização, se um usuário estiver utilizando uma estação, mas não estiver acessando o sistema, não há desperdício de processadores, apenas uma estação de baixo preço estará sendo ocupada.

Este é um modelo sugerido, que possibilita que os usuários usufruam de um grupo de processadores, e que seus terminais possam ser relativamente baratos, embora possam rodar programas de usuário. Os terminais podem ser uma máquina simples, com uma tela mapeada por mapas de bits e um mouse, podendo ser computadores pessoais ou estações de trabalho.

Por exemplo, no lugar de 50 estações de trabalho super-equipadas, a idéia é adquirir, por um preço equivalente, 50 processadores e 50 estações mais simples. Os processadores seriam organizados em grupos e constituídos por uma única placa conectada a uma rede, não precisariam de teclado, monitor ou mouse (fig. 4.4). Pode-se também não seguir o modelo, e utilizar os processadores em grupos de computadores pessoais ou estações de trabalho, que podem estar em locais físicos diferentes. Um dos objetivos é fazer com que a localização física dos processadores não modifique em nada a utilização do sistema. Os processadores de um grupo podem possuir diferentes arquiteturas, possibilitando que em um mesmo grupo funcionem máquinas com arquiteturas distintas.

FIGURA 4.4 - ARQUITETURA DO SISTEMA AMOEBA



FONTE: [TAN95]

Quando um usuário requisita um serviço, o sistema aloca um ou mais processadores para a execução do comando. Quando a tarefa é finalizada, os processadores são liberados, voltando ao grupo que pertencem, à espera de novas requisições, que podem vir de outros usuários. Se o número de tarefas solicitadas forem muitas, os processadores podem ser alocados em regime compartilhado, destinando-se novos processos aos processadores com menor carga.

O Amoeba possui também servidores especializados, que por questões de desempenho, sugere-se rodar em um processador separado. Os servidores fornecem serviços, que são as operações que poderão ser realizadas pelos usuários. Para tratar questões de tolerância à falhas, o Amoeba aloca diversos servidores para o mesmo serviço.

4.2.3 O MODELO DE SOFTWARE DO AMOEBA (*MICROKERNEL*)

O software do Amoeba possui dois módulos: um microkernel, que roda em todos os servidores do sistema, nos terminais, reconhecendo que eles são computadores e não apenas terminais simples e nos servidores especializados. Outro módulo é um conjunto de softwares para os servidores.

O microkernel do Amoeba tem quatro funções especiais, detalhadas a seguir.

4.2.3.1 GERENCIAR PROCESSOS E LINHAS DE CONTROLE (*THREADS*)

O Amoeba suporta o conceito de processo e de linhas de controle (*threads*) em um mesmo processo, utilizando o mesmo espaço de endereçamento. Um processo com uma única linha de controle possui um único espaço de endereçamento, um conjunto de registradores, um contador de programas e uma pilha. Em um processo com diversas linhas de controle, cada delas utiliza e compartilha o mesmo espaço de endereçamento.

Uma aplicação para diversas linhas de controle é um servidor de arquivos, onde cada requisição é atribuída a uma linha em particular. A linha de controle inicia a execução da requisição, ficando bloqueada até o acesso ao disco, depois continua o trabalho. O trabalho

do servidor é dividido pelas diversas linhas de controle e roda cada uma delas de forma sequencial, mesmo que precise ficar bloqueada aguardando entrada/saída.

É possível também a todas as linhas acessar um software compartilhado na *cache*. Para que não haja o acesso de duas linhas, ao mesmo tempo, à *cache* compartilhada, são utilizados mecanismos de sincronização, como, por exemplo, semáforos.

4.2.3.2 GERÊNCIA DE MEMÓRIA

O microkernel gerencia os blocos de memória que são alocados pelas linhas de controle para execução das requisições. Estes blocos são denominados segmentos.

Um processo precisa no mínimo de um segmento, ou pode precisar de vários, dependendo da tarefa a ser executada. Os segmentos podem ser alocados para dados, pilha ou qualquer coisa de que os processos necessitem. O sistema operacional não determina padrões para o uso dos segmentos.

4.2.3.3 COMUNICAÇÃO ENTRE PROCESSOS

O *kernel* pode controlar a comunicação entre processos de duas formas:

a) **Ponto a ponto**, quando um cliente envia uma mensagem a um servidor e fica bloqueado até receber resposta. A troca de mensagem/resposta é a base para todas as operações de comunicação;

b) **Em grupo**, quando uma mensagem é enviada de um ponto para diversos destinos. Através dos protocolos de comunicação; os processos podem comunicar-se em grupo, garantidos por mecanismos de tolerância à falhas.

4.2.3.4 GERENCIAMENTO DE ENTRADA/SAÍDA

Para cada dispositivo de entrada/saída, existe um *driver* ligado ao *kernel*. Esta ligação ocorre através de mensagens ponto a ponto. Cada *driver* controla todas as operações de entrada/saída do dispositivo correspondente. Por exemplo, se um processo como o servidor de arquivos precisa fazer uma comunicação com o disco, ele envia uma mensagem ao *driver* de disco e dele obtém resposta.

Ao cliente, esta comunicação com um *driver* do *kernel* é abstrata. Ele apenas faz sua solicitação e o sistema envia esta requisição ao lugar apropriado. O usuário não precisa se preocupar com estes detalhes.

Independente de um sistema de comunicação ser ponto a ponto ou em grupo, é necessário um protocolo especializado, denominado FLIP. Este protocolo pertence ao nível de rede e foi projetado para atender necessidades de um sistema distribuído

4.2.4 SERVIDORES DO AMOEBA

Os servidores existem para que os serviços oferecidos pelo sistema não sejam colocados diretamente dentro do *Kernel*, por questões de flexibilidade do sistema. Se o sistema de arquivos e outros serviços forem implementados em servidores, será permitido que diferentes versões sejam executadas simultaneamente, atendendo diferentes usuários e facilitando sua alteração.

O Amoeba é baseado no modelo cliente-servidor. Os clientes são escritos pelos próprios usuários e os servidores são responsabilidade do sistema, embora o Amoeba permita que os clientes escrevam seus próprios servidores.

O projeto do sistema é escrito utilizando o conceito de objeto. Os objetos podem ser arquivos, diretórios, segmentos de memória, as janelas de telas, processadores, discos ou *drives* de fita. Os objetos possuem dados e operações disponíveis sobre eles, e são gerenciados pelos servidores.

Quando um processo cria um objeto, o servidor que controla tal objeto devolve ao cliente uma “capacidade” desse objeto, protegida por criptografia. Qualquer uso posterior desse objeto depende de ser apresentada sua capacidade. As capacidades são informações necessárias para acesso, identificação e proteção dos objetos.

O servidor de arquivos do Amoeba, denominado de **servidor bala,** utiliza o conceito de arquivos imutáveis. Ele fornece aos usuários do sistema operações para criação, leitura e deleção de arquivos. Mas os arquivos, uma vez criados, não podem ser alterados, somente apagados ou rescritos.

O Amoeba tem também um servidor de diretórios, separado do servidor de arquivos. O servidor de diretórios é responsável pelo gerenciamento dos diretórios e nomes de caminho. Quando um processo deseja fazer uma leitura de um arquivo, ele solicita ao servidor de diretórios a busca do nome do caminho. Se for válido, o servidor retorna a capacidade do arquivo. A partir deste momento, qualquer operação realizada sobre este arquivo, necessitará apenas do servidor de arquivos, não mais do servidor de diretórios.

4.2.4.1 SERVIDOR-BALA

O Servidor-bala é o servidor de arquivos do Amoeba. O sistema de arquivos padrão do Amoeba contém três servidores: o **servidor-bala** que gerencia o armazenamento de informações, o **de diretório** que trata da identificação de arquivos e gerência dos diretórios e o **de replicação**, que trata a questão de replicação de arquivos.

Na operação de criação de um arquivo, o usuário utiliza a chamada de sistema *create*. O servidor bala atende a esta chamada devolvendo ao usuário uma capacidade, que poderá ser utilizada para posteriores acessos a este arquivo. Os arquivos são imutáveis, depois de criado, um arquivo não poderá ser modificado. Poderá ser apagado e um novo arquivo criado em seu lugar, mas não terá a mesma capacidade do antigo. Essa característica permite que o tamanho do arquivo seja definido antes de ser criado e que o arquivo seja armazenado de forma contígua, tanto no disco como na *cache*.

O princípio básico para criação dos arquivos é que o usuário crie um arquivo completo em sua memória, e depois, utilizando uma chamada remota a procedimento transmita o arquivo ao servidor-bala, que retorna uma capacidade ao cliente.

Modificações no arquivo são permitidas gerando, na verdade, a criação de um novo arquivo. O cliente envia a capacidade ao servidor, solicitando o arquivo a ser modificado. O arquivo é transmitido para a memória do cliente em uma única chamada remota a procedimento. O cliente faz as modificações no arquivo, envia novamente ao servidor que lhe retorna uma nova capacidade. Neste instante, o arquivo antigo pode ser destruído ou mantido a título de *backup*.

O servidor-bala também suporta clientes que tenham pouca memória para receber o arquivo inteiro. Para a leitura de arquivos, estes clientes podem solicitar uma única seção do arquivo, informando um deslocamento e um contador de *bytes*.

A escrita fica um pouco mais complicada neste caso, pois os arquivo são imutáveis. Para contornar este problema, o servidor-bala divide os arquivos em dois tipos: **não comprometidos**, que são os arquivos em processo de criação, e **comprometidos**, que são permanentes. O arquivos não comprometidos podem ser modificados, os comprometidos não. Os usuários utilizam uma chamada remota a procedimento para indicar se um arquivo deve ser comprometido imediatamente ou não.

Nos dois casos, uma cópia do arquivo é enviada ao servidor e é devolvida uma capacidade. Aos arquivos não comprometidos podem-se acrescentar informações. Quando todas as modificações terminarem, o arquivo pode passar à categoria de comprometido, tornado-se imutável.

As operações suportadas pelo sevidor-bala estão listadas na tabela 4.1.

TABELA 4.1 - CHAMADAS AO SERVIDOR-BALA

Create	Cria um novo arquivo, que pode já ser criado como comprometido ou não comprometido
Read	Permite ler determinado arquivo, todo ou parte dele
Size	Retorna o tamanho de determinado arquivo
Modify	Escreve *n bytes* por cima de um arquivo não comprometido
Insert	Insere *n bytes* em um arquivo não comprometido
Delete	Apaga *n bytes* de um arquivo não comprometido

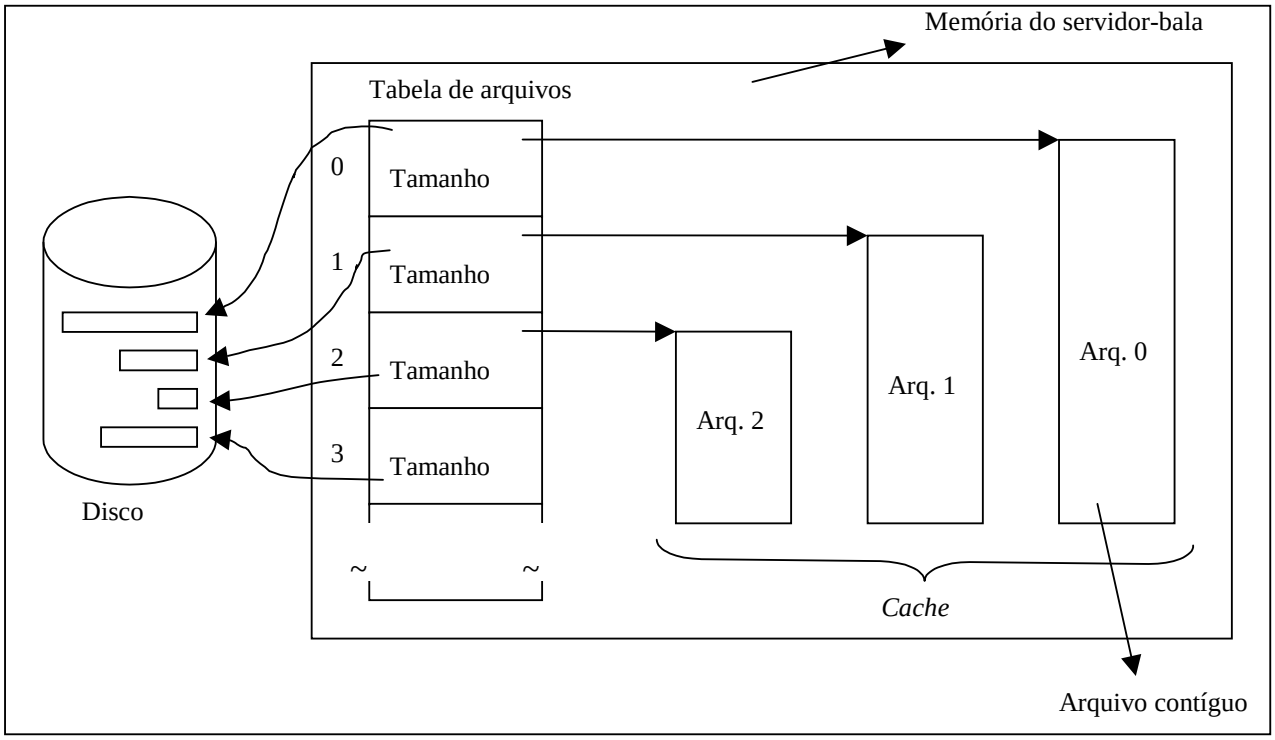
O procedimento *Create* fornece alguns dados que serão gravados em um novo arquivo, cuja capacidade é enviada de volta ao cliente que disparou a chamada. Um parâmetro do procedimento define se o arquivo será comprometido ou não comprometido. Se forem não comprometidos podem ser modificados ou acrescidos de informações.

A chamada *Read* permite somente ler arquivos comprometidos, no todo ou em parte. Esta chamada especifica o arquivo a ser lido através da capacidade. A apresentação da capacidade é a prova de que a operação é permitida. O servidor-bala não conhece a identidade do cliente e não faz nenhuma verificação que se baseie em tal identificação. A chamada *size* pega uma capacidade como parâmetro e retorna o tamanho do arquivo correspondente.

As três últimas chamadas da tabela 4.1 podem ser utilizadas somente em arquivos não comprometidos, permitindo que eles sejam modificados. O servidor-bala implementa também três chamadas especiais, disponíveis apenas ao administrador do sistema, possuidor de uma supercapacidade. Essas chamadas são utilizadas para transportar o conteúdo ca *cache* para o disco, compactar estes dados e recuperar sistemas de arquivos danificados.

O servidor-bala mantém uma tabela com uma entrada para cada arquivo, idêntica a tabela de nós-i do *Unix* (fig. 4.5). Esta tabela vai para a memória quando o servidor é inicializado e é mantida ali enquanto o servidor estiver executando.

FIGURA 4.5 - IMPLEMENTAÇÃO DO SERVIDOR-BALA



FONTE: [TAN95]

Cada entrada da tabela contém um ponteiro que dá o endereço do arquivo no disco, outro ponteiro que informa o endereço do arquivo na memória principal, um comprimento e algumas informações adicionais. Um ponteiro e um comprimento são suficientes para encontrar os dados de um arquivo, pois estes são armazenados de forma contígua, tanto no disco quanto na memória.

Na leitura de um arquivo, o cliente envia a capacidade ao servidor-bala, que extrai o número do objeto e o utiliza como índice para a tabela de arquivos. Na entrada correspondente existe um número, que é utilizado no campo *check* da capacidade, para

verificar sua validade. Se não for válida, a operação é finalizada com uma mensagem de erro. Se for válida o arquivo é buscado no disco e armazenado na *cache*.

Se um arquivo é criado e sua capacidade for perdida, nunca mais poderá ser acessado e nem excluído. Para prevenir isto, é utilizado um temporizador e definida a política de que, se um arquivo não comprometido não for acessado por 10 minutos, simplesmente é apagado e sua entrada na tabela fica disponível para outro arquivo. Se um novo arquivo utilizar esta entrada e a capacidade antiga for apresentada, o campo *check* indicará que a operação sobre o arquivo é inválida. A idéia é que os arquivos fiquem como não comprometidos por pouco tempo, enquanto estiverem sendo criados.

Para os arquivos não comprometidos existe um contador associado a sua entrada na tabela de arquivos, que é iniciado com um valor MAX-LIFETIME. Periodicamente um processo que executa em *background* faz a chamada *age*, que decrementa cada contador em uma unidade. Um arquivo cujo contador chegue a zero é destruído. Os espaços ocupados por ele na cache e na tabela são liberados.

Para evitar a exclusão de arquivos que estão em uso, existe outra chamada (*touch*) que se aplica a um arquivo específico, e faz com que o contador volte a ter o valor MAX-LIFETIME. Executa-se touch de tempos em tempos a todos os arquivos de um diretório.

4.2.4.2 SERVIDOR DE DIRETÓRIO

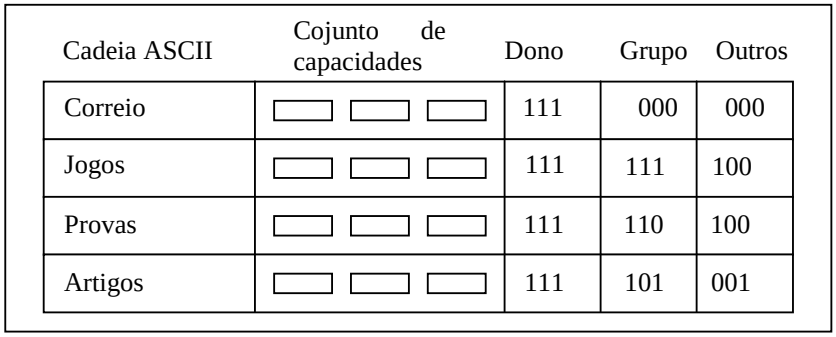
O servidor de diretório identifica os objetos e fornece procedimentos para mapear nomes ASCII para capacidades. Um diretório contém linhas e cada linha descreve um objeto e contém o nome do objeto e sua capacidade.

As operações permitidas pelo servidor de diretório são para criar e excluir diretórios, adicionar e apagar linhas e buscar nomes nos diretórios. Os diretórios são objetos que também possuem capacidades, podendo a capacidade de um diretório estar em uma linha de outro diretório, permitindo a formação de árvores hierárquicas.

Cada linha de um diretório pode conter mais que uma capacidade, que indica que existem cópias idênticas do objeto que podem ser gerenciadas por diferentes servidores (fig.

4.6). Na busca de um objeto, é fornecido todo o conjunto de capacidades. A busca é feita em uma capacidade de cada vez, até encontrar uma em que o servidor esteja disponível.

FIGURA 4.6 - DIRETÓRIO TÍPICO GERENCIADO PELO SERVIDOR DE DIRETÓRIO



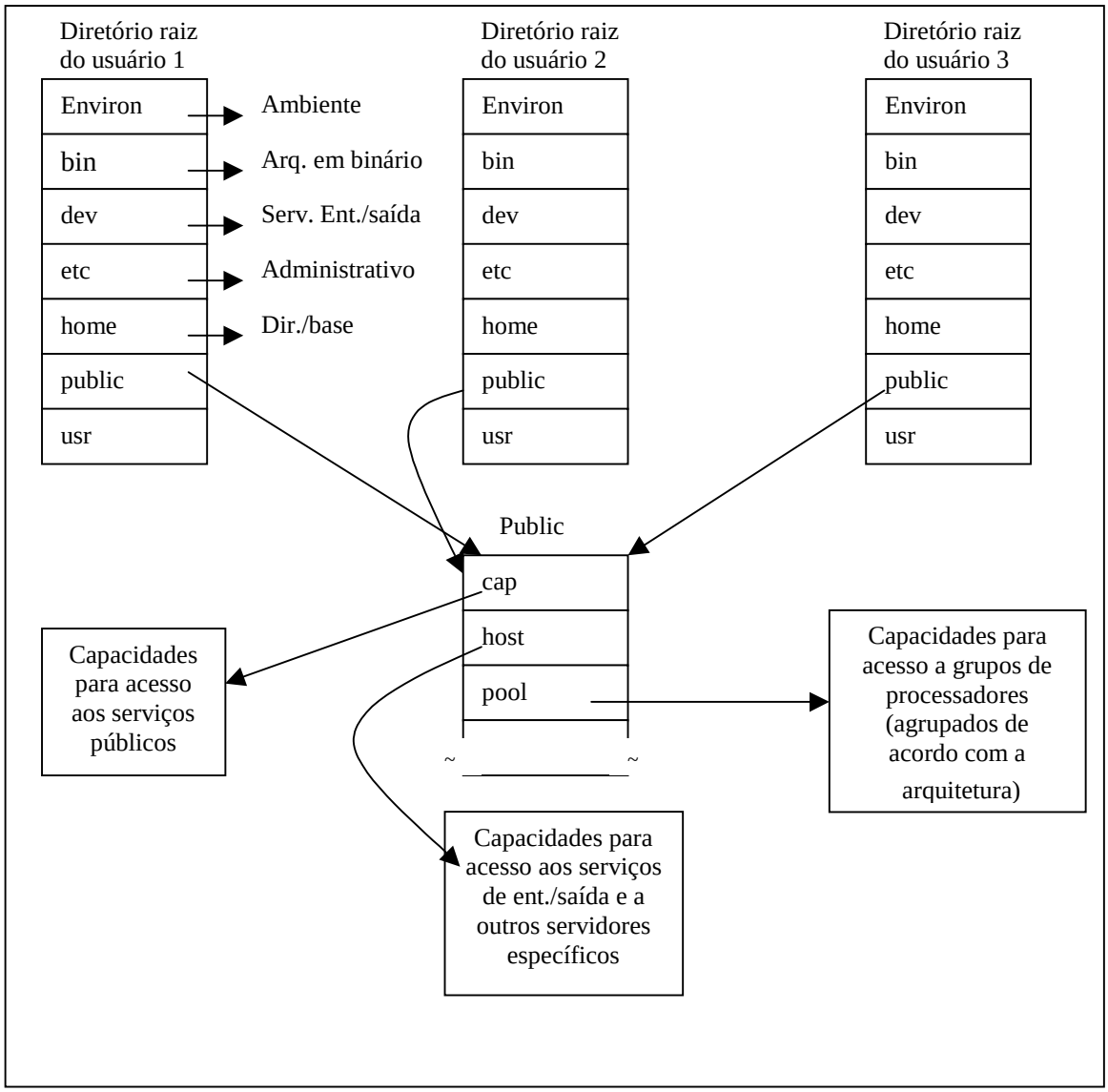
Cadeia ASCII	Cojunto de capacidades	Dono	Grupo	Outros
Correio		111	000	000
Jogos		111	111	100
Provas		111	110	100
Artigos		111	101	001

FONTE: [TAN95]

Cada linha do diretório pode conter também colunas que definem proteções e direitos de acesso ao objeto. A exemplo do esquema de proteção do *Unix*, pode-se ter uma coluna para definir as permissões do proprietário do objeto, uma para seu grupo e uma pública. O uso de capacidades que apontam para outros diretórios permitem a criação de árvores e o compartilhamento de arquivos. Um objeto pode ter capacidades em diferentes diretórios, pertencentes a diferentes usuários, podendo ter inclusive diferentes direitos de acesso.

No Amoeba, cada usuário tem seu próprio diretório, conforme fig. 4.7. Este pode conter capacidades não só para objetos privados do dono, como também para diversos diretórios públicos que são compartilhados. Um dos diretórios é denominado *public* e é a raiz da árvore pública compartilhada. O *home* é o diretório base do usuário.

FIGURA – 4.7 - HIERARQUIA DE DIRETÓRIO NO AMOEBA



FONTE: [TAN95]

Na tabela 4.2 estão descritas as chamadas que podem ser utilizadas para diretórios.

TABELA 2.2 - PRINCIPAIS CHAMADAS A UM SERVIDOR DE DIRETÓRIOS

Create	Cria um novo diretório
Delete	Apaga um diretório ou uma entrada no diretório
Append	Insere uma nova entrada em um diretório
Replace	Substitui uma entrada de um diretório
Lookup	Retorna o conjunto de capacidades correspondente as um nome específico
Getmasks	Retorna a máscara de direitos correspondente a uma entrada específica
Chmod	Modifica os bits de direito correspondentes a uma entrada existente no diretório

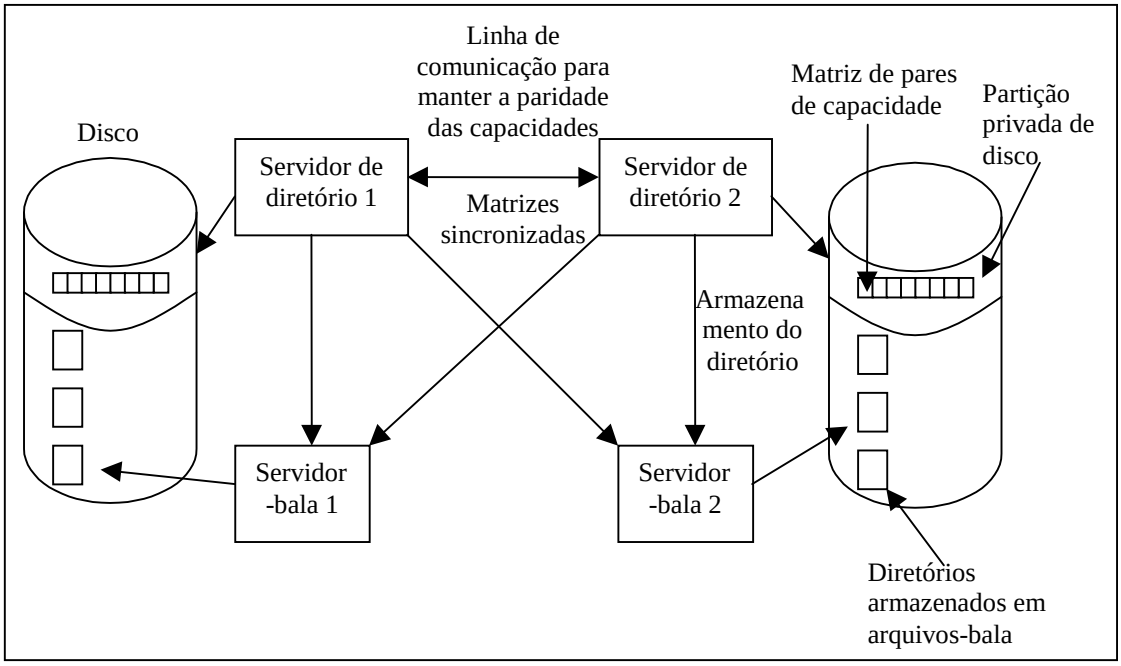
A implementação do servidor de diretórios do Amoeba é feita para ser tolerante a falhas. Sua estrutura básica é umas matriz de pares de capacidade, armazenada em uma parte privada do disco. Esta matriz precisa ser freqüentemente atualizada, por isso não usa o servidor-bala.

Na criação de um diretório, o número do objeto colocado em sua capacidade é um índice para esta matriz. Na apresentação de uma capacidade de diretório, o sistema verifica o número do objeto colocado na capacidade e busca o par de capacidade correspondente na matriz. As duas capacidades são para arquivos idênticos, armazenados em servidores-bala diferentes, cada um contendo o diretório e o campo de *check* para autenticar a capacidade do diretório.

Toda vez que um diretório é modificado, cria-se um novo arquivo-bala e as matrizes nas partições privadas do disco são alteradas. A segunda cópia é gerada automaticamente mais tarde, através de uma linha de controle que executa em *background*. Destroem-se então os diretórios antigos.

Os servidores de diretórios são em pares, cada um contendo sua própria matriz de pares de capacidade, em discos diferentes para prevenir problemas, caso a partição privada de um dos disco venha a falhar. Estes servidores se comunicam e mantém uma sincronização, embora possam funcionar isolados (fig. 4.8).

FIGURA 4.8 - PAR DE SERVIDORES DE DIRETÓRIO (OS DADOS SÃO ARMAZENADOS EM DUPLICIDADE EM DIFERENTES SERVIDORES-BALA)



FONTE: [TAN95]

4.2.4.3 SERVIDOR DE REPLICAÇÃO

O servidor de replicação é o responsável por criar réplicas dos objetos que são gerenciados pelo servidor de diretórios, através da **replicação preguiçosa**, ou seja, na criação de um objeto uma única cópia é criada. Depois o servidor de replicação pode ser chamado, criando quantas réplicas forem necessárias. O servidor de replicação fica executando em *background*, analisando partes específicas do servidor de diretórios. Ao encontrar entradas de diretórios que possuem determinado número de capacidades, mas que na verdade possua menos, entra em contato com os servidores adequados para criar réplicas do objeto.

O servidor de replicação também executa o mecanismo de *aging* e o de coleta de lixo utilizado pelo servidor-bala. Ele toca os objetos de tempos em tempos para impedir que sejam eliminados por *time-out*, e envia mensagens do *age* para os servidores, fazendo com que decrementem os contadores de objetos, eliminando os que atingirem zero.

4.2.4.4 SERVIDOR DE PROCESSAMENTO

O servidor de processamento auxilia na tomada de decisão quando o sistema recebe um comando do usuário e precisa decidir em qual processador específico o processo adequado irá executar, ou qual arquitetura será utilizada para o processo.

Cada servidor de processamento gerencia um ou mais grupos de processadores. Um diretório */pooldir* representa um grupo de processadores. Cada subdiretório é restrito a uma das arquiteturas de processador que são suportadas pelos sistema.

Quando um programa tiver que ser executado, ele é procurado em /*bin*. Se for um programa disponível para diversas arquiteturas, ele não será um único arquivo, mas um diretório contendo programas executáveis para cada arquitetura disponível. O *Shell* faz então uma chamada remota a procedimento para o servidor de processamento, solicitando que deixe disponível um processador de determinada arquitetura. Neste momento é feita uma análise do que existe disponível e faz-se uma seleção dos processadores que atendem os requisitos. A seleção específica fica por conta da verificação de memória disponível e estimativa do poder computacional que poderá ser dedicado ao programa. Estas informações sobre cada membro do grupo são controladas e calculadas pelo servidor de replicação.

4.2.4.5 SERVIDOR DE *BOOT*

O servidor de *boot* é utilizado principalmente para acrescentar um certo grau de tolerância a falhas ao sistema Amoeba. Ele mantém em seu arquivo de configuração entradas para os servidores que necessitam sobreviver a falhas, e faz uma verificação se os servidores que deveriam estar ativos estão realmente. Em caso de detecção de problemas, o servidor de *boot* procura alocar um novo grupo de processadores que inicializará uma cópia do servidor com problemas. Dessa forma, os serviços críticos são reconfigurados automaticamente em casos de falhas.

4.3 NETWORK FILE SYSTEM (NFS)

Desenvolvido pela Sun Microsystem, este sistema de arquivos foi projetado e implementado para uso em estações de trabalho que rodam *Unix*. Atualmente é utilizado por outros fabricantes, tanto em estações *Unix* quanto aquelas que rodam outros sistemas

operacionais. O NFS suporta sistemas heterogêneos e até mesmo com diferentes arquiteturas de hardware. Informações sobre o NFS, baseando-se em [TAN95], [SIL94] e [VAH96], estão descritas a seguir.

4.3.1 ARQUITETURA DO NFS

O NFS permite que cada máquina seja, ao mesmo tempo, cliente e servidor. Clientes e servidores podem compartilhar sistemas de arquivos comuns, e podem estar na mesma rede local ou rodando em uma rede de longa distância.

Os servidores de arquivos geralmente implementam sistemas hierárquicos, com um diretório raiz contendo subdiretórios e arquivos.

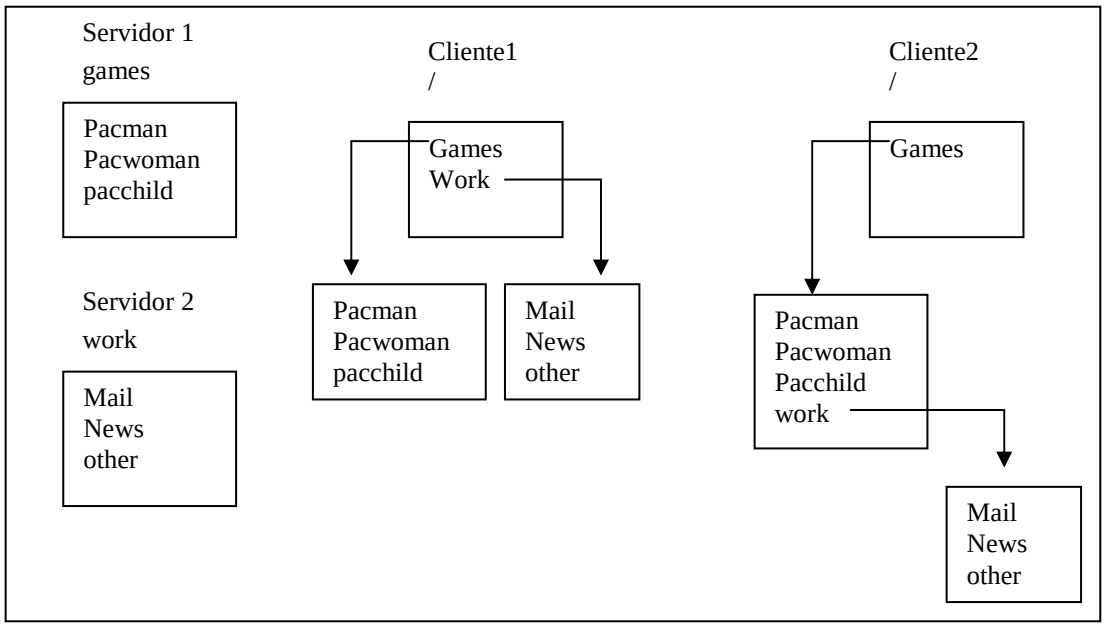
Os servidores exportam os seus diretórios, para que os clientes possam ter acesso remoto aos diretórios. Ao disponibilizar um diretório, todos os seus subdiretórios também ficam disponíveis. Os clientes podem exportar ou montar os sistemas de arquivos, aumentando o seu sistema de arquivos local.

A lista de diretórios que um servidor exporta é mantida no diretório */etc/exports*, assim, quando ocorre um *boot* no servidor, estes diretórios são exportados automaticamente.

Os diretórios exportados pelo servidor devem ser montados no sistema de arquivos do cliente. Nas estações de trabalho que possuírem disco, os clientes podem montar diretórios remotos em qualquer lugar do disco, criando um sistema de arquivos que é parcialmente local e parcialmente remoto. Nas estações de trabalho que não possuírem disco, os clientes podem montar os diretório em seu próprio diretório raiz, criando um sistema de arquivos totalmente suportado por um servidor remoto.

A fig. 4.9 mostra dois servidores de arquivos, um contendo o diretório *games*, e o outro um diretório *work*. Os dois clientes mostrados na figura montaram os dois diretórios em seus próprios sistemas de arquivos, só que colocando-os em posições diferentes. Apesar de não ter diferença o local onde um cliente monta o sistema de arquivos na sua hierarquia de arquivos, deve-se observar que clientes diferentes podem ter visões diferentes do mesmo diretório/subdiretório.

FIGURA 4.9 – CLIENTES DIFERENTES PODEM MONTAR OS SERVIDORES EM POSIÇÕES DIFERENTES



FONTE – [TAN95]

Para permitir compartilhamento de arquivos, dois ou mais clientes montam o mesmo diretório e comunicam-se através do compartilhamento dos arquivos de seus diretórios comuns. Os arquivos compartilhados são justamente aqueles pertencentes à hierarquia de diretórios de várias máquinas, podendo ser lidas e escritas de maneira usual.

4.3.2 PROTOCOLO

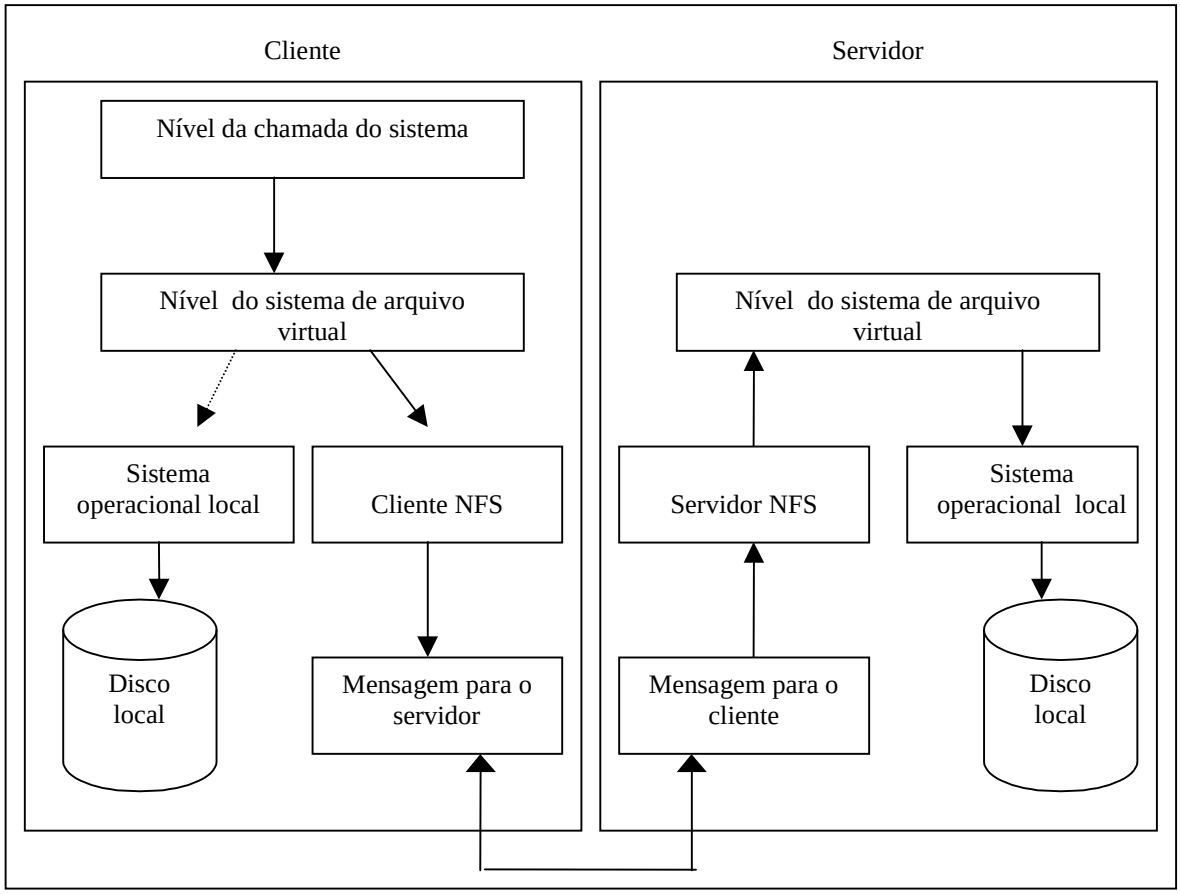
Protocolo é a definição de um conjunto de requisições enviadas por clientes a servidores, sendo que estas requisições são respondidas de volta ao cliente.

A implementação do protocolo NFS possui três níveis, conforme fig. 4.10. O **nível da chamada do sistema** é o nível mais alto e recebe as chamadas dos clientes. Ele verifica a sintaxe da chamada e valida seus parâmetros. Depois disso, ele chama o segundo nível.

O segundo e terceiro níveis são chamados de **nível do sistema de arquivo virtual (VFS) do cliente** e **nível do sistema de arquivo virtual (VFS) do servidor**. Este nível mantém uma tabela com uma entrada para cada um dos arquivos abertos. Cada entrada é denominada nó-V. O nó-V informa se o arquivo é local ou remoto. Se for remoto, ele fornece

todas as informações para acesso aos arquivos. Quando um cliente requisita a montagem de um diretório, depois de concedida a autorização, um nó-V é criado para o diretório remoto, e é então solicitado ao cliente que crie um nó-R (no-i remoto) em suas tabelas internas, para armazenar as informações da autorização. Dessa forma, um nó-V conterá um ponteiro para um nó-R no código do cliente ou um ponteiro para um nó-i no sistema de arquivos local. A partir de um nó-V é possível verificar se um arquivo é remoto ou local, e se for remoto, encontrar sua autorização de manipulação.

FIGURA 4.10 ESTRUTURA EM CAMADAS DO NFS



FONTE: [TAN95]

O NFS define dois protocolos cliente-servidor, descritos abaixo.

4.3.2.1 PROTOCOLO NFS PARA MONTAGEM DE ARQUIVOS

O cliente solicita permissão para a montagem de um diretório especificado, enviando um nome de caminho. O local da montagem não precisa ser especificado na mensagem. Se o

caminho for válido e o diretório for exportado pelo servidor, este envia uma mensagem ao cliente denominada autorização para manipulação de arquivo.

A mensagem contém campos que identificam o tipo de sistema de arquivos, o disco onde se encontra, o número do nó-i do diretório e informações sobre segurança. As chamadas subsequentes para leitura e escrita no diretório montado utilizam estas informações.

4.3.2.2 PROTOCOLO NFS PARA ACESSO A ARQUIVOS E DIRETÓRIOS

Para leitura de um arquivo, o cliente envia uma mensagem ao servidor, contendo o nome do arquivo e uma solicitação para examiná-lo, recebendo de volta uma autorização para manipular o arquivo, extraindo dele a estrutura que o identifica.

Esta operação não copia informações nas tabelas internas do sistema. A chamada *read* contém a informação enviada pelo servidor sobre o arquivo a ser lido, a posição inicial de leitura dentro do arquivo e o número de bytes a serem lidos. As próprias mensagens contém todas as informações necessárias para a realização da operação.

Não se perde informação a respeito de arquivos abertos, quando um servidor está com problemas, já que não há informações desse tipo armazenadas no servidor. Um servidor que não mantém informações sobre os arquivo abertos é chamado servidor sem informação de estado.

Como os servidores do NFS não sabem quais os arquivos estão abertos, não é possível aplicar a semântica exata *Unix*. Por exemplo, no *Unix*, um arquivo pode ser aberto e bloqueado para que outro processo não tenha acesso a ele. Quando o arquivo é fechado, é automaticamente desbloqueado.

O NFS utiliza o mecanismo de proteção *Unix*, com os bits *rwx* para o proprietário, para o grupo e para os demais usuários. Cada mensagem de solicitação ao servidor contém a identificação do usuário e do grupo que o chamou. Pode ser utilizado um sistema público de criptografia por questões de segurança, na hora de validar o cliente e o servidor em cada solicitação e resposta.

Chaves utilizadas para autenticação e mais algumas informações são mantidas pelo NIS (*Network Information Service*). É uma espécie de página amarela e armazena pares (chave, valor). Se a chave é fornecida, ele retorna o valor correspondente. O NIS controla a criptografia das chaves, mapeamento dos nomes de usuários para senhas criptografadas e mapeamento de nomes de máquinas para os endereços da rede.

4.3.3 MONTAGEM E LOCALIZAÇÃO DE OBJETOS

A chamada *mount* permite montar um sistema de arquivos remoto. Algumas informações são necessárias para a montagem do sistema de arquivos remoto, dentre elas, a especificação do diretório remoto e o diretório local onde o diretório remoto deve ser montado. Pelo nome do diretório remoto o *mount* descobre em que máquina ele se encontra. Após fazer um contato com a máquina, solicita autorização para a manipulação do diretório remoto. Se o diretório existir e estiver disponível para a esta operação, o servidor devolve uma autorização ao cliente, e a partir dessa autorização o *kernel* recebe as informações necessárias para a montagem. O *kernel* constrói um nó-V para o diretório remoto e solicita que o cliente crie um nó-R em suas tabelas internas.

Para a abertura de um arquivo remoto, o *kernel* verifica em qual diretório o arquivo vai ser montado, e se é de fato um arquivo remoto. No diretório dos nós-V é encontrado o nó-R correspondente, e neste momento o controle é passado para o código NFS que corresponde à abertura de arquivo. Este código procura o restante do nome do caminho no diretório montado e obtém uma autorização para manipulação do diretório. Este código prepara então uma entrada de um nó-R em suas tabelas internas e envia informações de volta ao nível VFS, que por sua vez coloca em suas tabelas um nó-V que aponta para o nó-R desse arquivo.

Após a abertura de um arquivo, o processo que requisitou esta chamada recebe um descritor de arquivo para o arquivo remoto, que poderá ser utilizado em operações posteriores, onde o nível VFS localiza o nó-V correspondente e, a partir dele, determina se o arquivo é local ou remoto e se aponta para um nó-R ou nó-I.

4.3.4 PERFORMANCE X CONSISTÊNCIA

Uma das maneira de melhorar a performance do sistema é aplicada na transferência de dados. As transferências entre cliente e servidor são feitas utilizando um número

relativamente grande de bytes, normalmente 8.192, mesmo que a quantidade necessária de bytes a ser transferidos seja menor. Quando o cliente obtém dados do servidor, após sua camada VFS ter recebido 8 kb, ele automaticamente requisita a leitura do próximo bloco. Esta característica é denominada leitura antecipada e melhora consideravelmente a performance.

Em operações de escrita, quando os dados a serem enviados ao servidor forem menor que 8kb, eles ficam acumulados localmente, e só quando a quantidade necessária for atingida são realmente enviados. Somente quando um arquivo é fechado, todos os seus dados são transferidos imediatamente ao servidor, mesmo não tendo atingido os 8kb.

Outra técnica para a melhora da performance é a utilização de *cache* no servidor, para evitar acessos a disco. Isto é totalmente transparente ao cliente. O cliente mantém duas *caches*, uma para atributos dos arquivos (nós-i) e outra para os dados dos arquivos.

A *cache* no cliente melhora a performance, mas causa problemas de consistência de dados. Uma das formas que o NFS utiliza para reduzir o problema é a colocação de um temporizador associado a cada bloco da *cache*. Quando ele expira, a entrada correspondente ao bloco é descartada. É comum utilizar 3 segundos para blocos de dados e 30 segundos para blocos de diretórios. Além disso, toda vez que um arquivo da *cache* é aberto, uma mensagem é enviada ao servidor para saber quando o arquivo foi modificado pela última vez. Se a última modificação ocorreu depois que a cópia local foi para a *cache*, a cópia da *cache* é invalidada e uma nova cópia é feita a partir do servidor. Quando o temporizador de uma das *caches* expira (30 segundos), todos os blocos modificados são enviados para o servidor.

O conjunto de semânticas de acesso a arquivos não é muito bem definido, pois não trata todas as possíveis falhas que possam vir a ocorrer.

5 DESENVOLVIMENTO DO TRABALHO

Um sistema de arquivos, como visto anteriormente, é a parte mais visível de um sistema operacional. Para o projeto e implementação de um gerenciador de arquivos para ambiente distribuído, aspectos referentes ao compartilhamento, transparência e possíveis operações a serem realizadas devem estar bem definidos, para possibilitar acesso aos arquivos de forma transparente e para garantir a consistência dos dados no sistema.

5.1 ESPECIFICAÇÃO DOS REQUISITOS

Levando em consideração os tópicos abordados nas seções anteriores, o protótipo de gerenciador de arquivos deverá implementar alguns dos conceitos e técnicas estudados.

Para a proteção dos arquivos, o protótipo deverá utilizar, à exemplo do *Unix*, os bits de proteção *rwx*, para o proprietário do arquivo, para seu grupo e para os outros usuários.

A transparência de localização garante que o usuário não precisará se preocupar com a localização física do arquivo. O protótipo deverá ser capaz de eleger um servidor que possua espaço para armazenar um arquivo a ser criado.

Para facilitar a organização lógica dos dados, o protótipo deverá implementar a estrutura de diretórios em árvore. Cada usuário terá um diretório pessoal e poderá criar outros diretórios a partir deste, formando uma hierarquia em árvore.

Para especificar nomes de arquivos, o protótipo deverá permitir que seja utilizada a denominação absoluta ou relativa, já que o sistema terá como base um diretório raiz e também implementa o conceito de diretório corrente.

Como semântica de compartilhamento, os arquivos abertos para escrita serão bloqueados para outros usuários e, para abrir um arquivo para leitura, uma verificação para saber se o arquivo já não está aberto para escrita é feita.

Como não será utilizado *cache* de blocos de dados, a consistência de informações deve ser mantida através de controles com informações a respeito apenas da localização do arquivo, através de tabelas em memória, tanto do cliente como do servidor. Atualizações em

arquivos são feitas diretamente no servidor e, no momento da modificação ou inclusão de novas informações, o arquivo fica bloqueado para demais usuários.

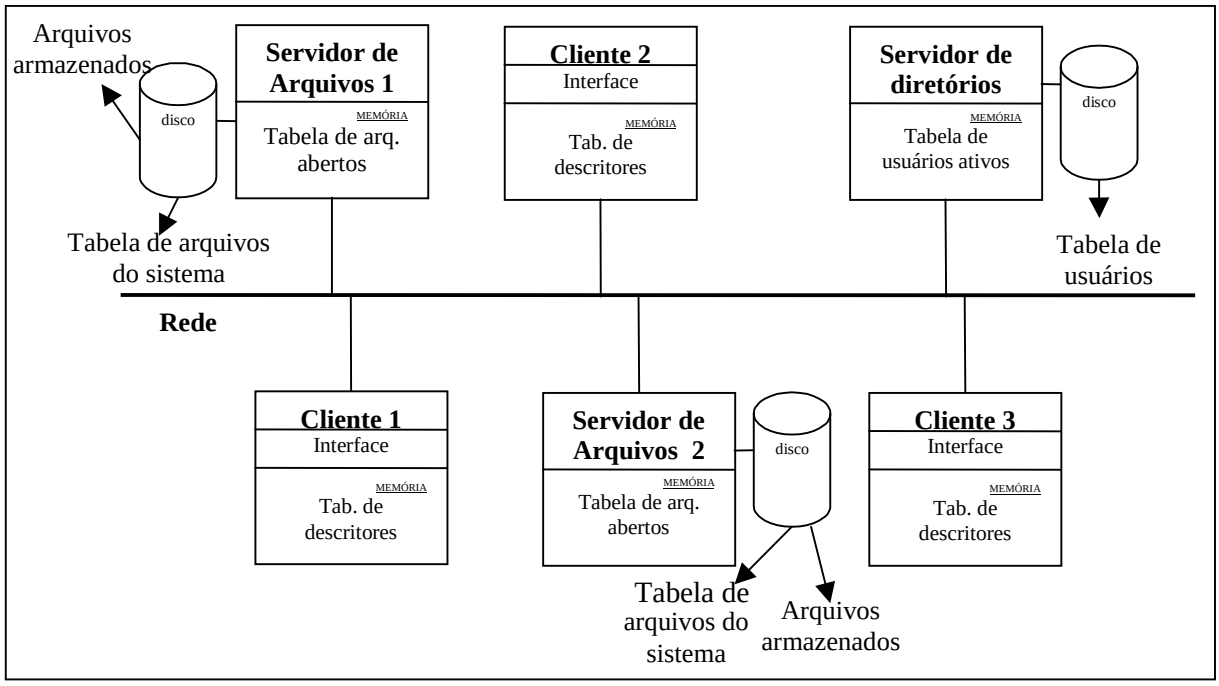
5.2 ESTRUTURA DO PROTÓTIPO DE GERENCIADOR DE ARQUIVOS

Visando atender os requisitos estabelecidos anteriormente, optou-se pelo desenvolvimento do protótipo de gerenciador de arquivos de forma a dar acesso aos arquivos localmente ou remotamente, podendo os clientes e servidores estarem na mesma rede local ou em redes de longa distância. Uma máquina pode ser cliente e servidor ao mesmo tempo. Podem haver diversos servidores de arquivos, um servidor de diretórios e vários clientes (fig. 5.1)

O protótipo de gerenciador de arquivos desenvolvido tem uma estrutura do tipo cliente/servidor. Assim, tem-se alguns componentes que são responsáveis por atender requisições sobre arquivos. Os clientes utilizarão os serviços do sistema.

Os arquivos ficam armazenados nos servidores de arquivos, em qualquer ponto da rede e são acessados através da implementação de algumas primitivas que permitem criar, abrir, escrever ou inserir dados em arquivos, bem como fechar e excluir. Estas primitivas estarão disponíveis aos usuários através da interface de um módulo que executará em todas as estações clientes.

FIGURA 5.1 - ESTRUTURA GERAL DO PROTÓTIPO DE GERENCIADOR DE ARQUIVOS



A manipulação de diretórios também é possível e é controlada através de um servidor de diretórios. Este servidor mantém uma estrutura que armazena o diretório de cada usuário, criando uma estrutura de diretórios em árvore. A estrutura de diretórios possui um diretório raiz chamado "/". A partir deste diretório, estão os diretórios de cada usuário e estes podem conter subdiretórios. O servidor de diretórios faz a localização de arquivos solicitados através das chamadas de sistema, retornando a localização do arquivo requerido. Algumas primitivas foram definidas para criar, excluir, listar e mudar de diretório (fig. 5.7, 5.8, 5.9 e 5.10).

A estrutura que o servidor de diretórios mantém na memória, com informações de cada usuário que está conectado, possui também um campo denominado diretório corrente. Quando um usuário acessa o gerenciador de arquivos, por padrão, o diretório corrente é o /*home/user*, sendo que *user* é o seu diretório pessoal. A partir do momento que ele utiliza o comando de mudança de diretório, este campo é atualizado para o diretório especificado.

Para efeitos de performance, o módulo cliente mantém uma estrutura em memória, que contém informações sobre os arquivos abertos. Essas informações apontam para uma entrada na tabela de arquivos abertos do servidor de arquivos. Uma operação realizada sobre um arquivo aberto não necessita mais do procedimento de localização. Para realizar operações de leitura, escrita e inserção no arquivo é necessário primeiramente abri-lo.

Se um cliente abre um arquivo para escrita ou inserção, este arquivo fica bloqueado para os demais clientes, para manter uma integridade de dados.

5.2.1 PROTEÇÃO AOS ARQUIVOS

Na estrutura geral de arquivos, mantida pelos servidores de arquivos, ficam diversas informações como proprietário do arquivo, nome do arquivo, etc. Entre estas informações, um campo define as permissões para acesso ao arquivo. Conforme o método *Unix* de proteção, este campo contém 3 conjuntos de 3 bits cada um, totalizando 9 bits. O primeiro bit de cada conjunto especifica permissão para leitura, o segundo para escrita e o terceiro para execução do arquivo. O primeiro conjunto refere-se ao usuário, o segundo ao seu grupo e o terceiro a todos os usuários do sistema que não fazem parte do grupo do dono.

5.3 MÓDULOS DO PROTÓTIPO E ESTRUTURAS DE DADOS

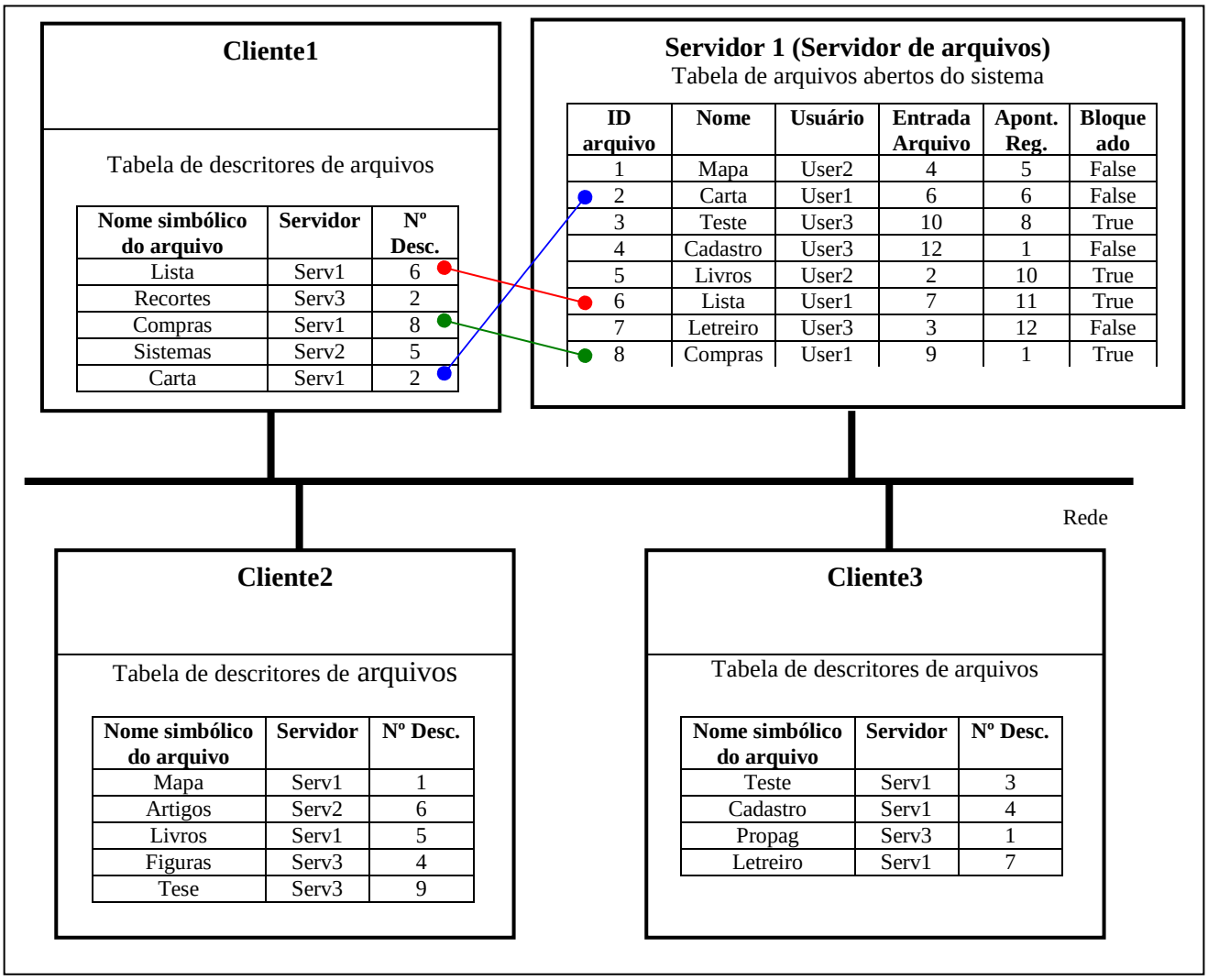
O protótipo do gerenciador de arquivos para ambiente distribuído é constituído por três módulos: um cliente que roda em todas as estações de trabalho, um servidor de arquivos, que pode rodar em uma ou mais máquinas do sistema, e um servidor de diretórios, que executará em uma das máquinas do sistema.

5.3.1 MÓDULO CLIENTE

Faz interface com o usuário. Aceita comandos para operações em arquivos e diretórios. Quando o usuário estabelece conexão, o módulo cliente mantém na memória uma tabela de descritores de arquivos. Toda vez que o usuário utiliza o comando de abertura de arquivo e for bem sucedido, é adicionada uma entrada na tabela de descritores de arquivos, e mantida lá até que o arquivo seja fechado. Quando o usuário utilizar o comando para fechamento do arquivo, a entrada é retirada da tabela. Enquanto o arquivo estiver aberto, sua entrada na tabela de descritores de arquivos é utilizada para qualquer operação posterior realizada sobre este arquivo.

Cada entrada na tabela de descritores de arquivos contém uma posição, que é na verdade uma entrada para uma outra tabela, a tabela de arquivos abertos do sistema (localizada nos servidores de arquivos, mantida na memória) (fig. 5.2).

FIGURA 5.2 - ESTRUTURAS DE DADOS DO PROTÓTIPO



Se o usuário abrir o arquivo no modo *Read/Write* (para leitura e escrita), este arquivo fica marcado como bloqueado na tabela de arquivos abertos do sistema, para que nenhum outro usuário possa abrir ou apagar o arquivo.

O cliente mantém também uma tabela de servidores, com o número IP e o tipo de serviço prestado pelo servidor (fig. 5.3).

FIGURA 5.3 - TABELA DE SERVIDORES

Servidor	Nº IP	Serviço
Serv1	200.19.218.65	Servidor de Arquivos
Serv2	200.17.278.54	Servidor de Arquivos
Serv3	200.18.653.21	Servidor de Diretórios

5.3.2 MÓDULO SERVIDOR DE ARQUIVOS

Recebe chamadas para operações em arquivos. As primitivas implementadas pelo servidor de arquivos estão descritas na tabela 5.1.

TABELA 5.1 – PRIMITIVAS PARA OPERAÇÕES EM ARQUIVOS

OPEN	Abertura de um arquivo
READ	Leitura de um arquivo
WRITE	Escrita em um arquivo
DELETE	Exclusão de um arquivo
CLOSE	Fechamento de um arquivo

O servidor de arquivos mantém uma tabela de todos os arquivos abertos (de todos os usuários). Esta tabela contém algumas informações sobre os arquivos, além de uma posição que aponta para uma tabela geral de arquivos (fig. 5.4), gravada em disco. Esta tabela é controlada pelos servidores de arquivos e contém, entre outras coisas, o endereço no disco de todos os arquivos, abertos ou não.

FIGURA 5.4 - TABELA GERAL DE ARQUIVOS DO SISTEMA (EM DISCO)

ID	Nome	Usuário	Endereço	Permissões
0				
1	Letras	User2	End1	111100100
2	Livros	User2	End2	111100100
3	Letreiro	User3	End3	111100100
4	Mapa	User2	End4	111100100
5	Matérias	User1	End5	111100100
6	Carta	User1	End6	111100100
7	Lista	User1	End7	111110100
8	Provas	User3	End8	111100100
9	Compras	User1	End9	111100100
10	Teste	User3	End10	111100100
11	Bordas	User2	End11	111111110
12	Cadastro	User3	End12	111100100
13	Produto	User2	En13	111100100
14	Marcas	User1	End14	111111110

A comunicação entre servidor de arquivos e cliente é feita obedecendo um protocolo pré-definido, para cada uma das operações. O cliente requisita um serviço e aguarda resposta do servidor de arquivos. O servidor recebe a chamada, identifica, chama os procedimentos correspondentes àquele serviço e devolve ao cliente uma mensagem de erro ou o resultado da operação.

A máquina em que o servidor de arquivos executará fica disponível para ser também cliente ou servidor de diretórios. Na criação de um arquivo, o servidor de diretórios elege, obedecendo alguns critérios, um dos servidores de arquivos disponíveis no sistema para armazenar o arquivo, guardando essa informação em suas estruturas, para localização posterior do arquivo, sem que o cliente precise ficar a par dessa informação.

5.3.3 MÓDULO SERVIDOR DE DIRETÓRIOS

Recebe chamadas para operações em diretórios. Implementa as primitivas que estão descritas na tabela 5.2.

TABELA 5.2 – PRIMITIVAS PARA OPERAÇÕES EM DIRETÓRIOS

MKDIR	Cria um novo diretório
CHDIR	Muda de diretório
RMDIR	Apaga um diretório existente
LIST	Lista o conteúdo de um diretório

O servidor de diretórios mantém uma tabela em disco com o nome do usuário, senha, grupo do usuário e um apontador para seu diretório pessoal (fig. 5.5). Quando o usuário se conecta, o servidor de diretórios valida o cliente, comparando seu nome e senha com esta tabela. A partir deste momento, é dada entrada em uma tabela na memória (fig. 5.6), que contém os mesmos dados, mas somente dos usuários que estão conectados.

Além de controlar a manipulação de diretórios, o servidor de diretórios faz a localização de arquivos dentro dos diretórios, retornando um apontador para o arquivo na tabela do sistema e identificando o servidor em que se encontra; ou uma mensagem de erro, caso o arquivo não seja encontrado.

FIGURA 5.5 - TABELA DE USUÁRIOS COM APONTADOR PARA O DIRETÓRIO PESSOAL DE CADA UM (DISCO)

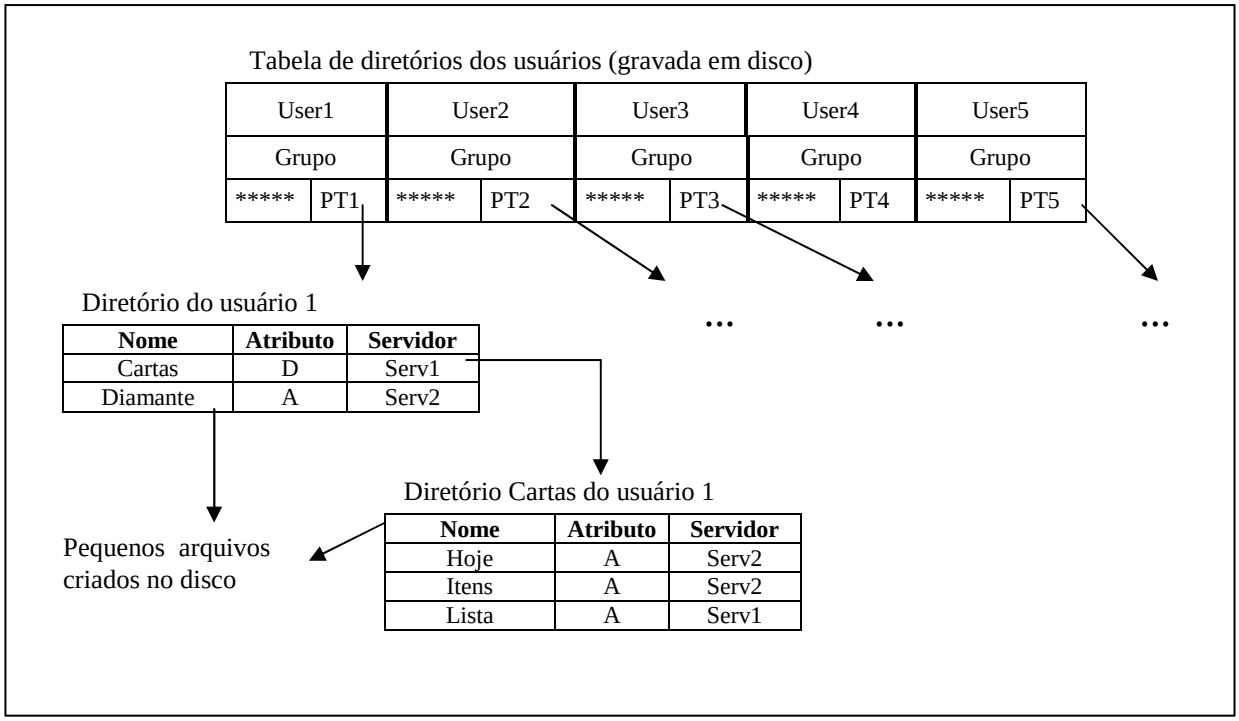


FIGURA 5.6 - TABELA DOS USUÁRIOS ATIVOS (MEMÓRIA)

Usuário	**Grupo**	**Ponteiro Diretório**	**Diret. Corrente**
User1	Gp1	PT1	/Home
User2	Gp1	PT2	/Home/User2

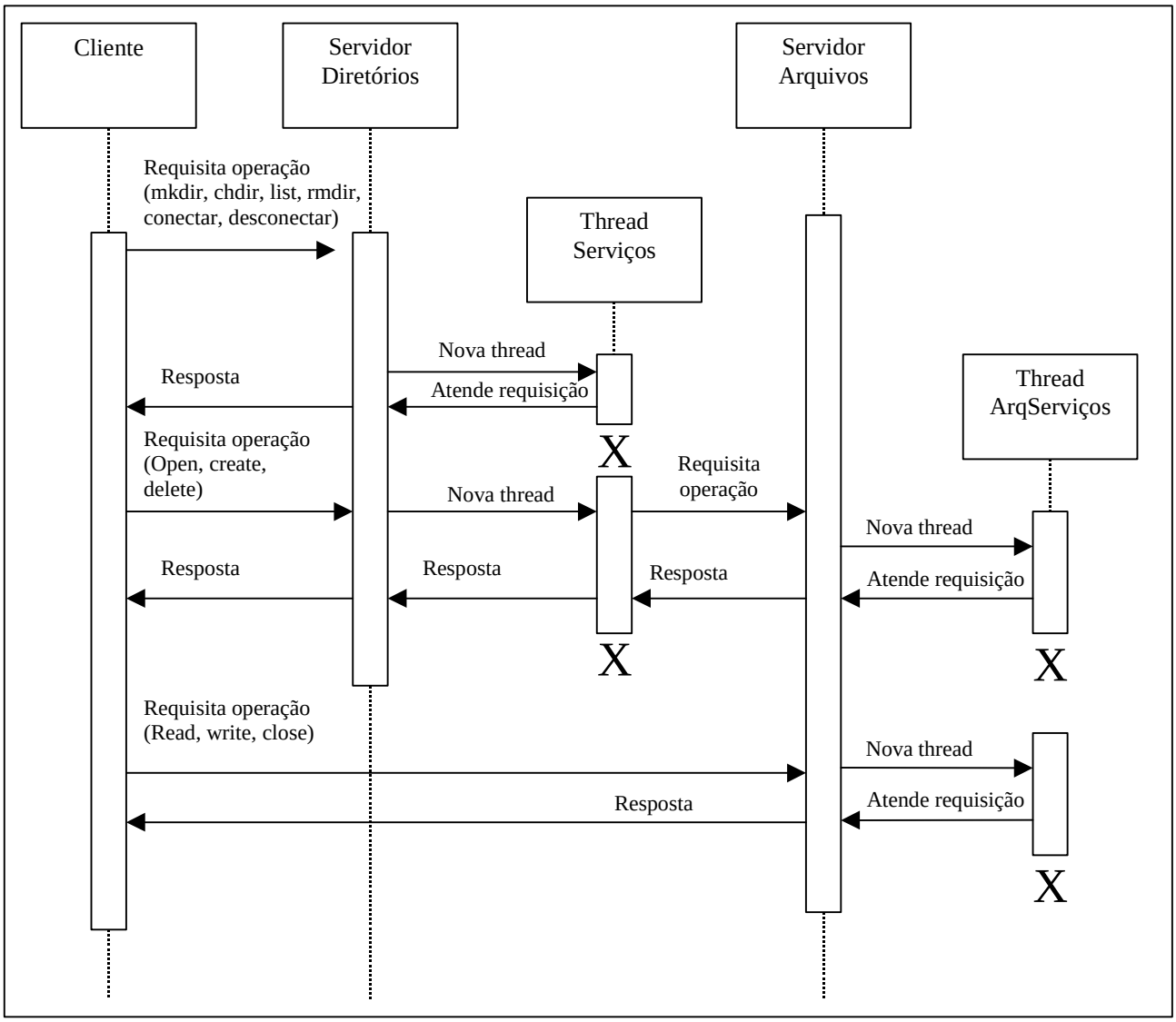
Inicialmente, quando o usuário se conecta ao sistema, por *default* o diretório corrente dele é /*home/user*, sendo que *user* é o seu diretório pessoal. Ao efetuar o comando *Chdir* ele estará modificando o diretório corrente para o especificado no comando.

A entidade servidor de diretórios existe para que a localização e validação dos arquivos possa ficar centralizada, evitando a redundância destes controles nos servidores de arquivos, onde o cliente teria que buscar um determinado arquivo tentando nos diversos servidores de arquivos, ou, no mínimo, consultando uma tabela em um deles para obter a localização do arquivo. Isso geraria problemas na hora de atualizar estas tabelas, já que teria que estar em todos os servidores de arquivos.

Utilizando a modelagem de objetos através da UML (Unified Modeling Language), a

fig. 5.7 mostra um diagrama de sequência com o funcionamento geral do protótipo.

FIGURA 5.7 – DIAGRAMA DE SEQUÊNCIA DO FUNCIONAMENTO GERAL DO PROTÓTIPO



5.4 ESPECIFICAÇÃO DAS PRIMITIVAS DE COMANDOS

Para disponibilizar operações sobre arquivos e diretórios, o protótipo implementa primitivas de comandos, que serão descritas e detalhadas a seguir.

5.4.1 OPERAÇÕES SOBRE DIRETÓRIOS

Para a manipulação de diretórios, o gerenciador de arquivos permite a criação, remoção, listagem e mudança de diretório. Para isso, o cliente utiliza os comandos descritos na tabela 5.2

5.4.1.1 CRIAÇÃO DE DIRETÓRIOS - *MKDIR*

O comando ***mkdir*** cria um novo diretório, caso não exista. O formato desta primitiva está descrito no quadro 5.1.

QUADRO 5.1 – SINTAXE DA PRIMITIVA MKDIR

MKDIR ([NOME DO CAMINHO], <NOVO DIRETÓRIO>)

O usuário pode informar o nome do caminho onde deseja criar o novo diretório. Caso não informe, o sistema considera que o novo diretório será criado no diretório corrente do usuário.

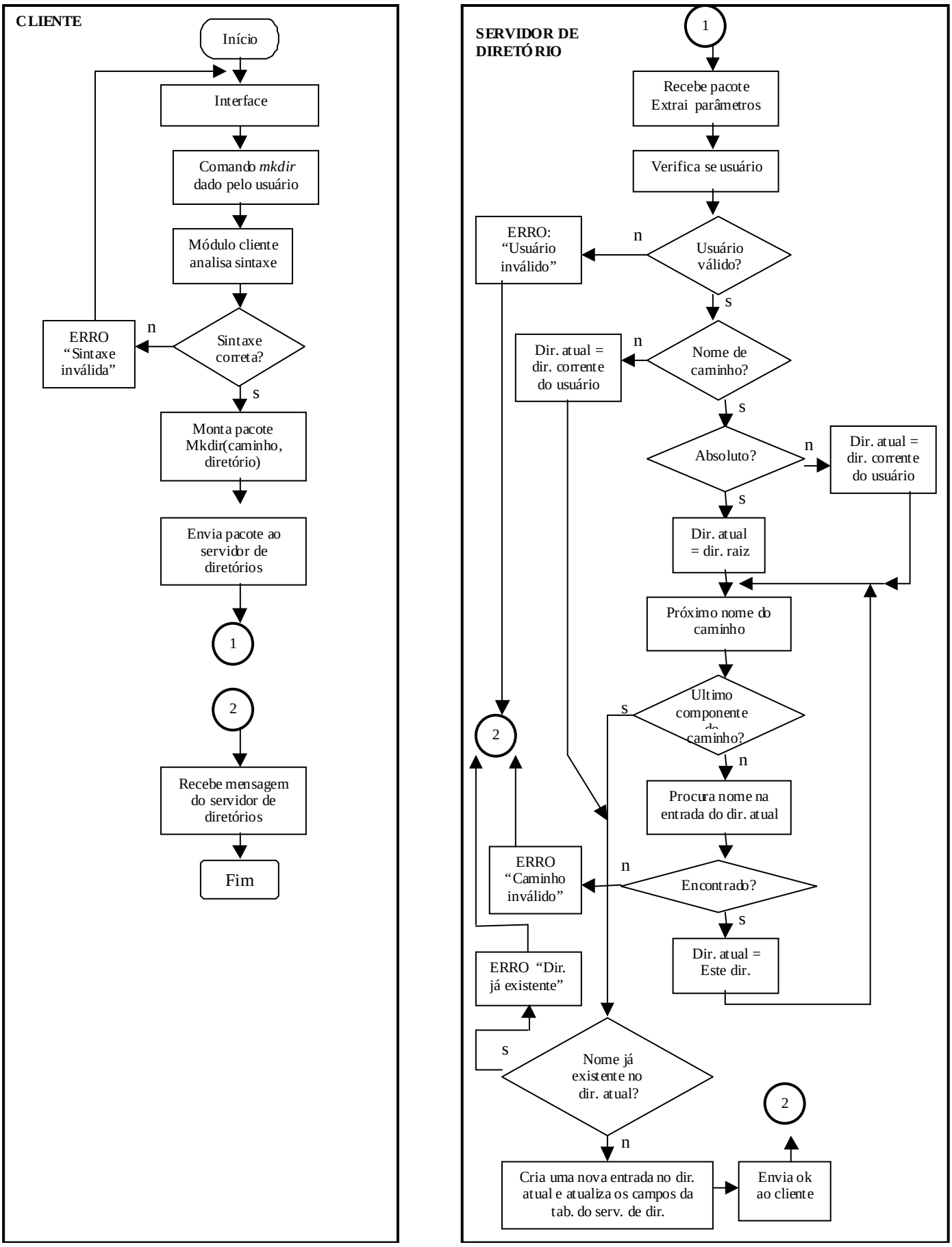
Ao receber o comando *mkdir*, o servidor de diretórios verifica se existe um nome de caminho. Se existir e for absoluto, o servidor de diretórios faz uma pesquisa a partir do diretório raiz, verificando se o caminho existe. Em caso negativo, uma mensagem informando caminho inválido é enviada. Sendo positivo, é verificado se já não existe uma entrada para o diretório que está sendo criado. Não existindo, é então criada uma nova entrada no diretório indicado pelo nome do caminho, sendo a operação finalizada com sucesso. Caso contrário, uma mensagem de “diretório já existente” é enviada.

Se o nome do caminho indicado for relativo, o processo para identificação do diretório indicado pelo nome do caminho é o mesmo, só que não inicia no diretório raiz, e sim a partir do diretório corrente do usuário.

Se não for informado nome de caminho, o servidor de diretórios verifica se já existe uma entrada para o diretório no próprio diretório corrente, criando-o com sucesso ou enviando uma mensagem que informa que o diretório já existe.

O funcionamento da primitiva *mkdir* está especificado na fig. 5.8.

FIGURA 5.8 - ESPECIFICAÇÃO DO COMANDO *MKDIR*



5.4.1.2 MUDANÇA DE DIRETÓRIO - *CHDIR*

O comando ***chdir*** modifica o diretório corrente do usuário para o diretório especificado. O formato da primitiva *chdir* está no quadro 5.2.

QUADRO 5.2 – SINTAXE DA PRIMITIVA *CHDIR*

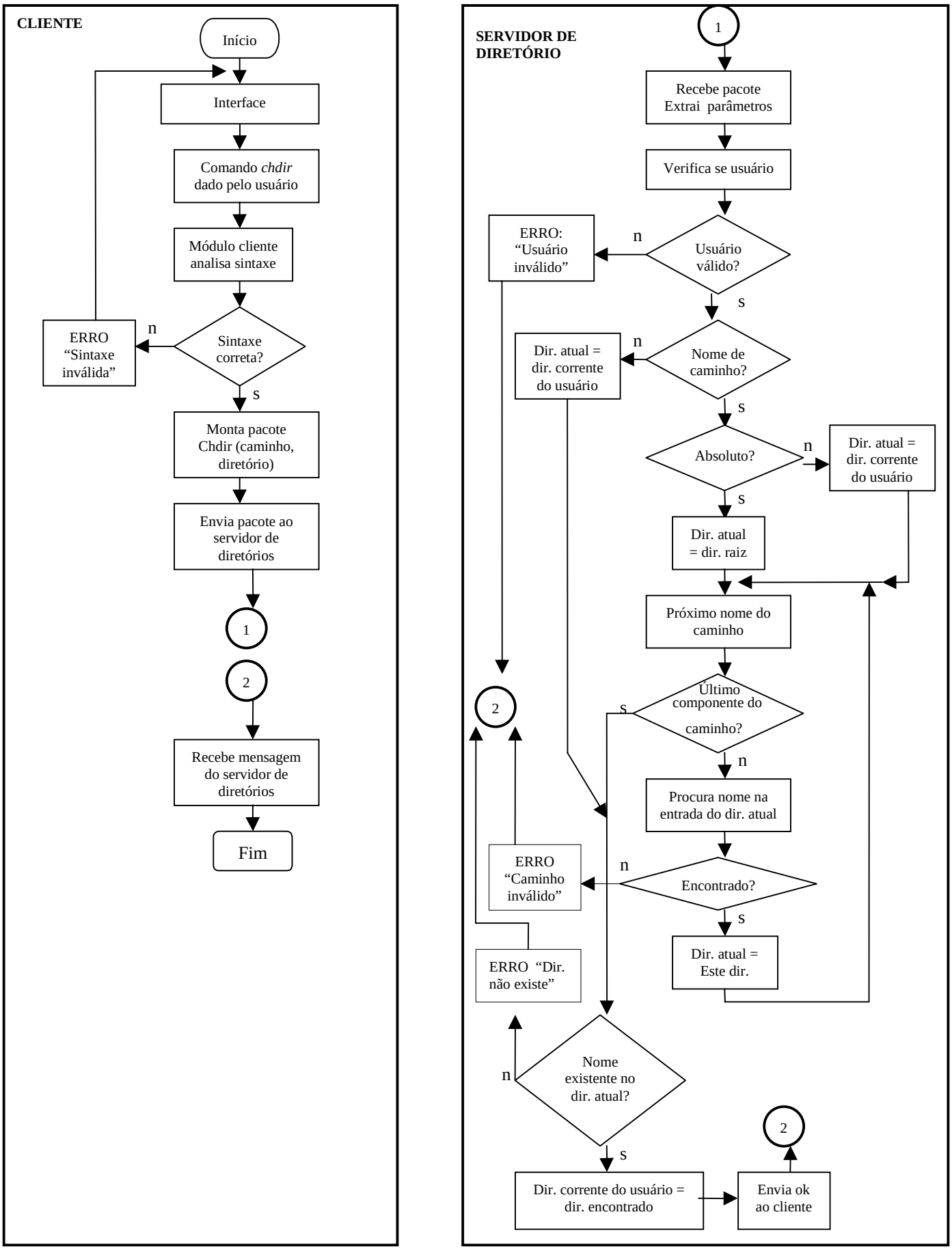
CHDIR ([NOME DO CAMINHO], <NOME DO DIRETÓRIO>)

Ao receber o comando *chdir*, o servidor de diretórios verifica o nome do diretório para onde deseja mudar. Caso o nome for através de um caminho absoluto, o servidor de diretórios faz uma pesquisa a partir do diretório raiz, verificando se o caminho existe. Em caso negativo, uma mensagem informando “caminho inválido” é enviada. Sendo positivo, o diretório corrente do usuário passa a ser o diretório especificado pelo caminho. A fig. 5.9 demonstra o funcionamento da primitiva *chdir*.

Se o nome for através de um caminho relativo, o processo para identificação do diretório indicado pelo nome do caminho é o mesmo, só que não inicia no diretório raiz, e sim a partir do diretório corrente do usuário.

Se for informado somente o nome, o servidor de diretórios procura o diretório especificado pelo comando no próprio diretório corrente do usuário.

FIGURA 5.9 – ESPECIFICAÇÃO DA PRIMITIVA *CHDIR*



5.4.1.3 EXCLUSÃO DE DIRETÓRIO - *RMDIR*

A chamada ***rmdir*** permite eliminar um diretório informado como parâmetro. O formato da primitiva *rmdir* está descrito no quadro 5.3.

QUADRO 5.3 – SINTAXE DA PRIMITIVA *RMDIR*

RMDIR ([NOME DO CAMINHO] ,<NOME DO DIRETÓRIO>)

O usuário pode informar o nome do caminho para o diretório a ser eliminado. Caso não informe, o sistema considera que o diretório especificado fica no diretório corrente do usuário.

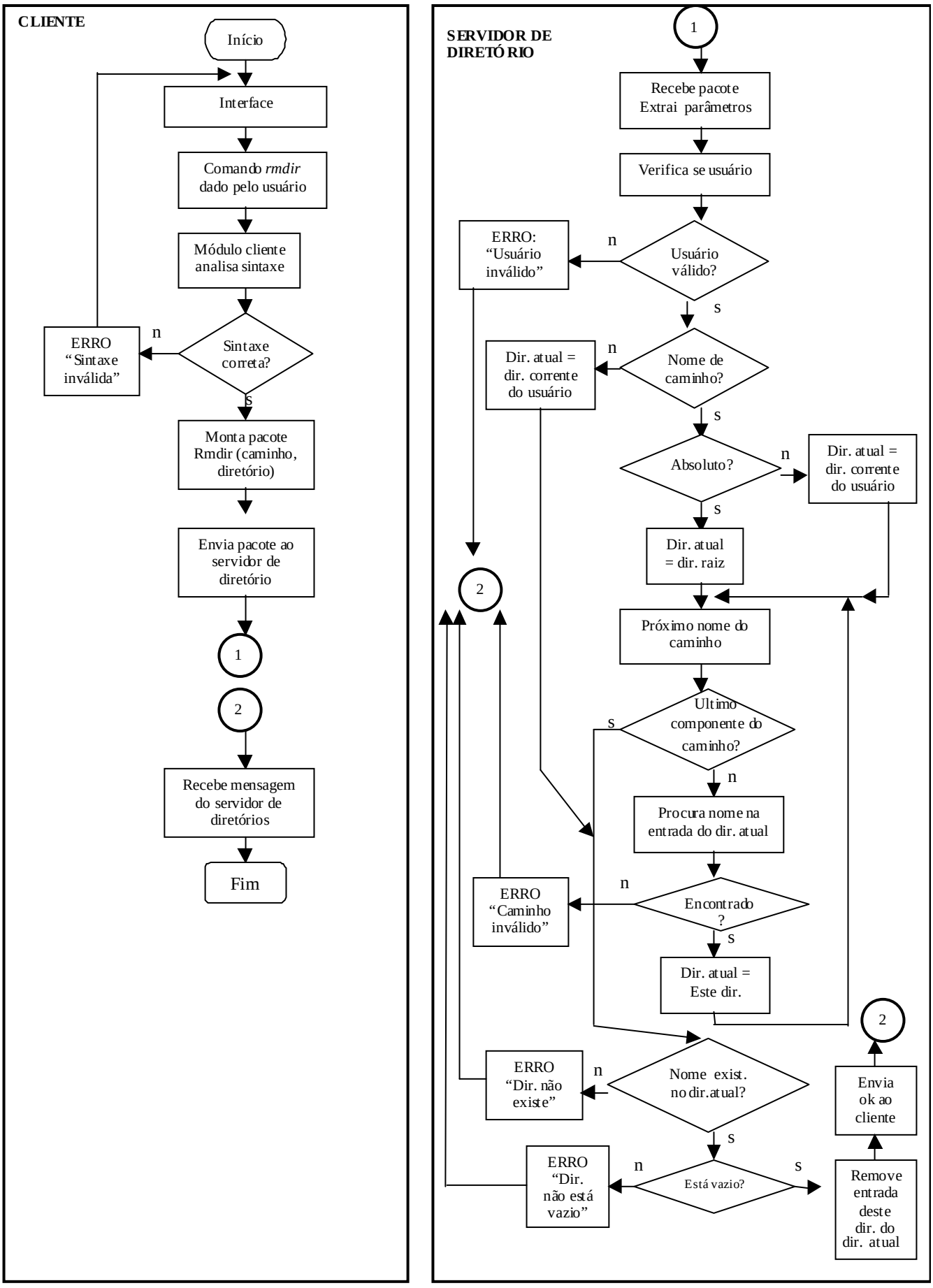
Ao receber o comando *rmdir*, o servidor de diretórios verifica se há nome de caminho. Caso existir e for um nome de caminho absoluto, o servidor de diretórios faz uma pesquisa a partir do diretório raiz, verificando se o caminho existe. Em caso negativo, uma mensagem informando “caminho inválido” é enviada. Sendo positivo, se o diretório estiver vazio, é eliminado. Se não estiver vazio, uma mensagem de erro retorna ao usuário informando “diretório não está vazio”.

Se o nome do caminho indicado for relativo, o processo para identificação do diretório indicado pelo nome do caminho é o mesmo, só que não inicia no diretório raiz, e sim a partir do diretório corrente do usuário.

Se não for informado nome de caminho, o servidor de diretórios procura o diretório especificado pelo comando no próprio diretório corrente do usuário.

A fig. 5.10 mostra o funcionamento da primitiva *rmdir*.

FIGURA 5.10 – ESPECIFICAÇÃO DA PRIMITIVA *RMDIR*



5.4.1.4 LISTAGEM DE DIRETÓRIO - *LIST*

A chamada ***list*** permite listar o conteúdo de um determinado diretório. O quadro 5.4 mostra o formato da primitiva *list*.

QUADRO 5.4 – SINTAXE DA PRIMITIVA *LIST*

LIST ([NOME DO CAMINHO])

O usuário pode informar o nome do caminho para o diretório que deseja listar. Caso não informe, o sistema considera que o diretório especificado fica no diretório corrente do usuário.

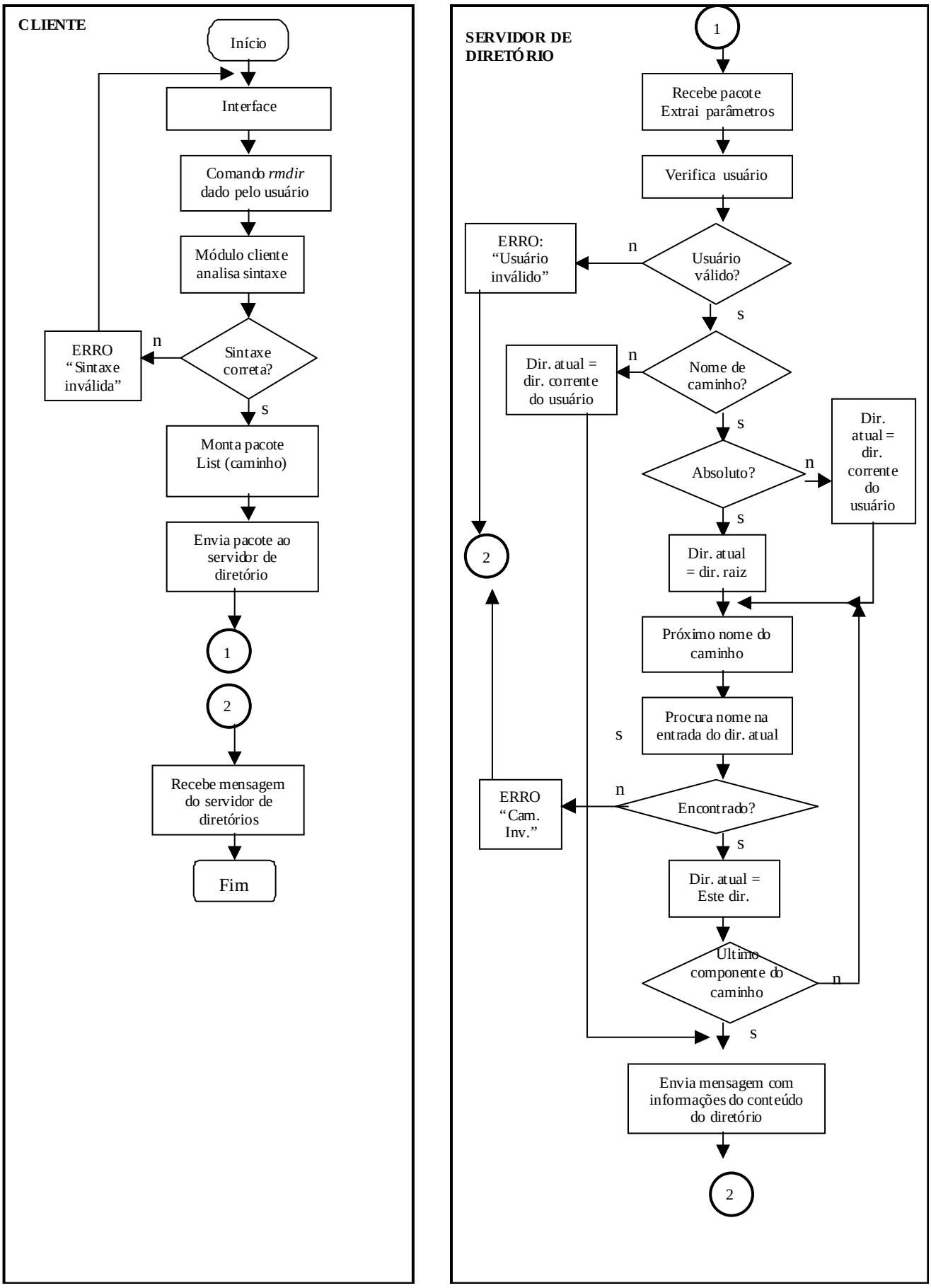
Ao receber o comando *list*, o servidor de diretórios verifica se há nome de caminho. Caso existir e for um nome de caminho absoluto, o servidor de diretórios faz uma pesquisa a partir do diretório raiz, verificando se o caminho existe. Em caso negativo, uma mensagem informando “caminho inválido” é enviada. Sendo positivo, uma listagem de todo o conteúdo deste diretório é enviada ao cliente.

Se o nome do caminho indicado for relativo, o processo para identificação do diretório indicado pelo nome do caminho é o mesmo, só que não inicia no diretório raiz, e sim a partir do diretório corrente do usuário.

Se não for informado nome de caminho, o servidor de diretório faz uma listagem do diretório corrente do usuário.

A fig. 5.11 especifica o funcionamento da primitiva *list*.

FIGURA 5.11 – ESPECIFICAÇÃO DA PRIMITIVA *LIST*



5.4.2 OPERAÇÕES SOBRE ARQUIVOS

A manipulação de arquivos é possível através da implementação de chamadas para criação, remoção, leitura e escrita dos arquivos. Para operações de leitura e escrita, é necessário que o arquivo seja aberto primeiramente e dependendo do modo como é aberto, fica bloqueado aos demais usuários. Ao ser fechado, arquivos bloqueados são então liberados para uso.

5.4.2.1 ABERTURA DE ARQUIVO - *OPEN*

O comando ***open*** disponibiliza o arquivo a determinado usuário para leitura ou escrita. Ao abrir um arquivo, o usuário deve informar o modo que deseja abrir (*r* para leitura e *rw* para leitura e escrita), caso contrário, o sistema utiliza o modo padrão (*r*). O quadro 5.5 mostra o formato da primitiva *open.*

QUADRO 5.5 – SINTAXE DA PRIMITIVA *OPEN*

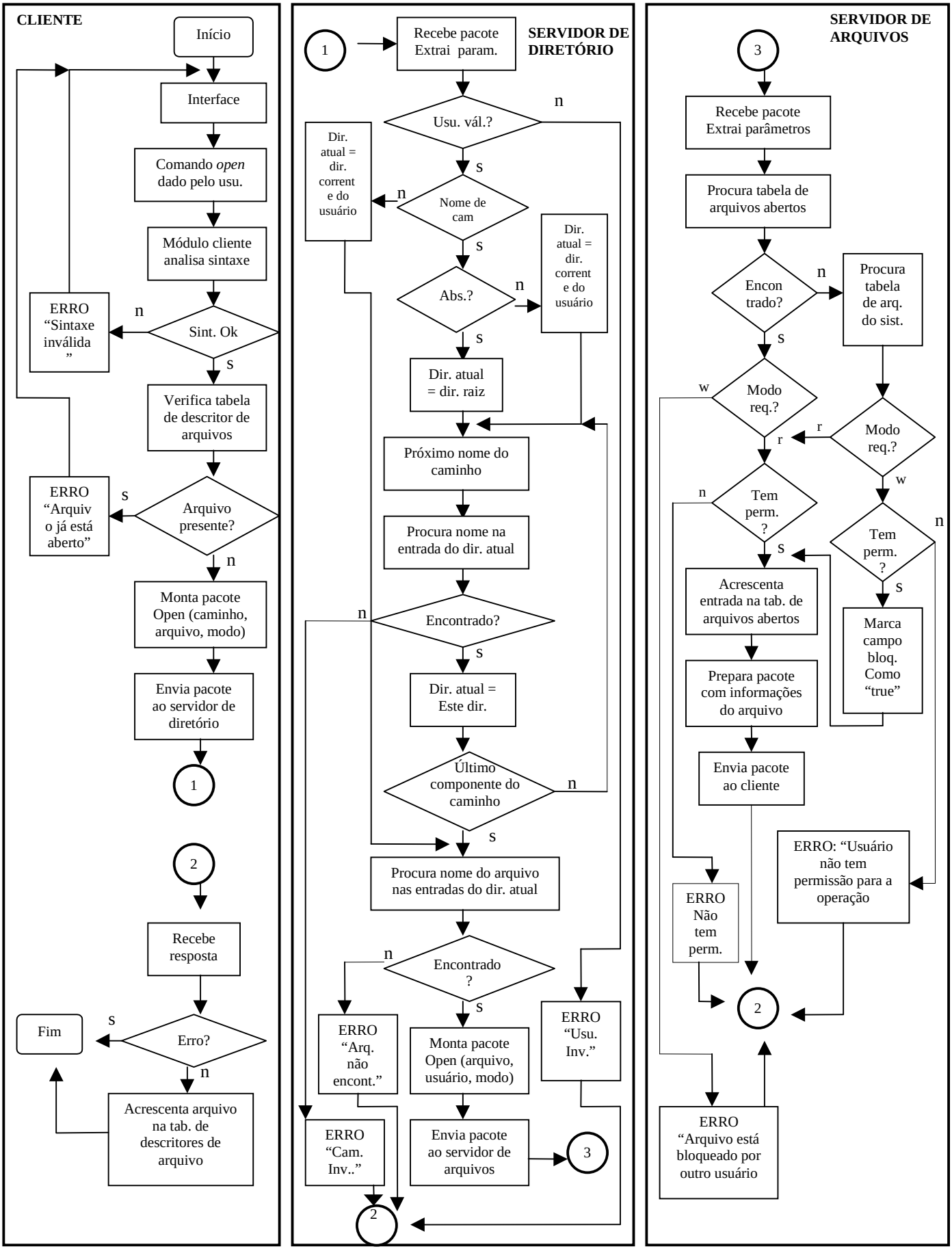
OPEN ([NOME DE CAMINHO], ARQUIVO, [MODO])

Se um nome de caminho for informado, este será validado e se for correto, a procura pelo arquivo será feita neste diretório. Todo o processo de validação de nome de caminho, absoluto e relativo é idêntico ao relatado nas operações anteriores.

Se um nome de caminho não for informado, o diretório corrente do usuário será percorrido em busca do arquivo. A abertura do arquivo, se bem sucedida, resulta na criação de uma entrada na tabela de descritores de arquivos do cliente que solicitou a operação. Esta entrada conterá todas as informações para posterior localização do arquivo, seu nome de caminho absoluto e inclusive o servidor de arquivos onde está localizado fisicamente. Qualquer operação de leitura, escrita ou fechamento feita posteriormente, só precisa utilizar estas informações e contatar diretamente o servidor de arquivos específico.

O modo padrão é leitura (*r*), a não ser que um modo explícito seja especificado juntamente com o comando. Encontrado o arquivo a ser aberto, uma verificação nas permissões é feita, para saber se é permitido ao usuário que está tentando abrir o arquivo realizar esta operação. A fig. 5.12 mostra a especificação da primitiva *open*.

FIGURA 5.12 – ESPECIFICAÇÃO DA PRIMITIVA *OPEN*



5.4.2.2 CRIAÇÃO DE ARQUIVO - CREATE

O comando ***create*** permite criar um arquivo. O quadro 5.6 mostra o formato da primitiva *create*.

QUADRO 5.6 – SINTAXE DA PRIMITIVA *CREATE*

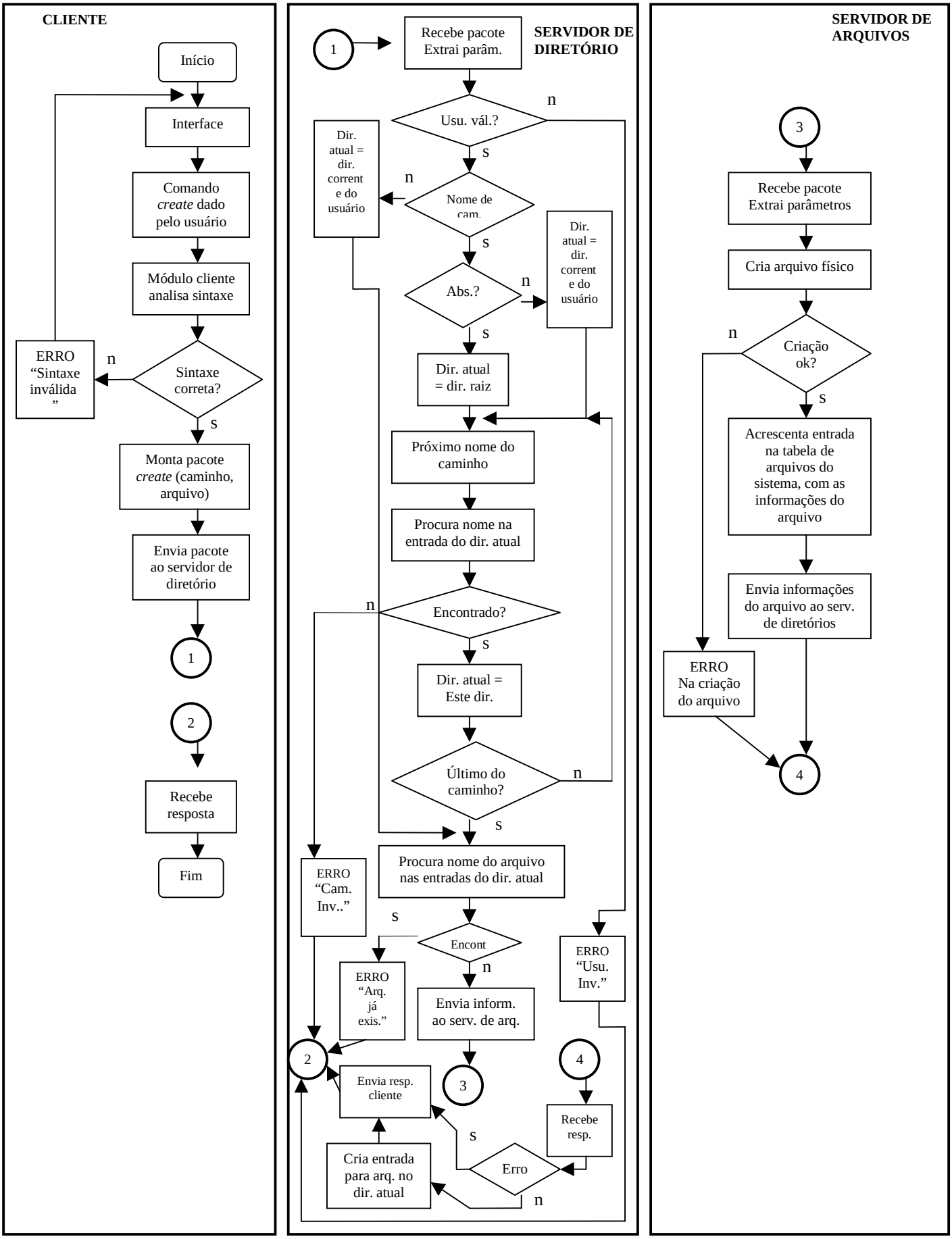
CREATE([NOME DE CAMINHO], ARQUIVO)

Se um nome de caminho for informado, este será validado e se for correto, todo o processo de validação de nome de caminho, absoluto e relativo é idêntico ao relatado nas operações anteriores. Se o diretório especificado for válido, uma verificação para constatar se o arquivo já existe é feita. Se não existir, é criado no diretório especificado, e acrescentado na tabela de arquivos do sistema.

Se um nome de caminho não for informado, o diretório corrente do usuário será utilizado para verificação de existência do arquivo e criação, caso não exista.

A fig. 5.13 mostra a especificação da primitiva *create*.

FIGURA 5.13 – ESPECIFICAÇÃO DA PRIMITIVA *CREATE*



5.4.2.3 LEITURA DE ARQUIVO - *READ*

O comando ***read*** disponibiliza a leitura integral ou parcial do arquivo especificado. O quadro 5.7 mostra o formato da primitiva *read*.

QUADRO 5.7 – SINTAXE DA PRIMITIVA *READ*

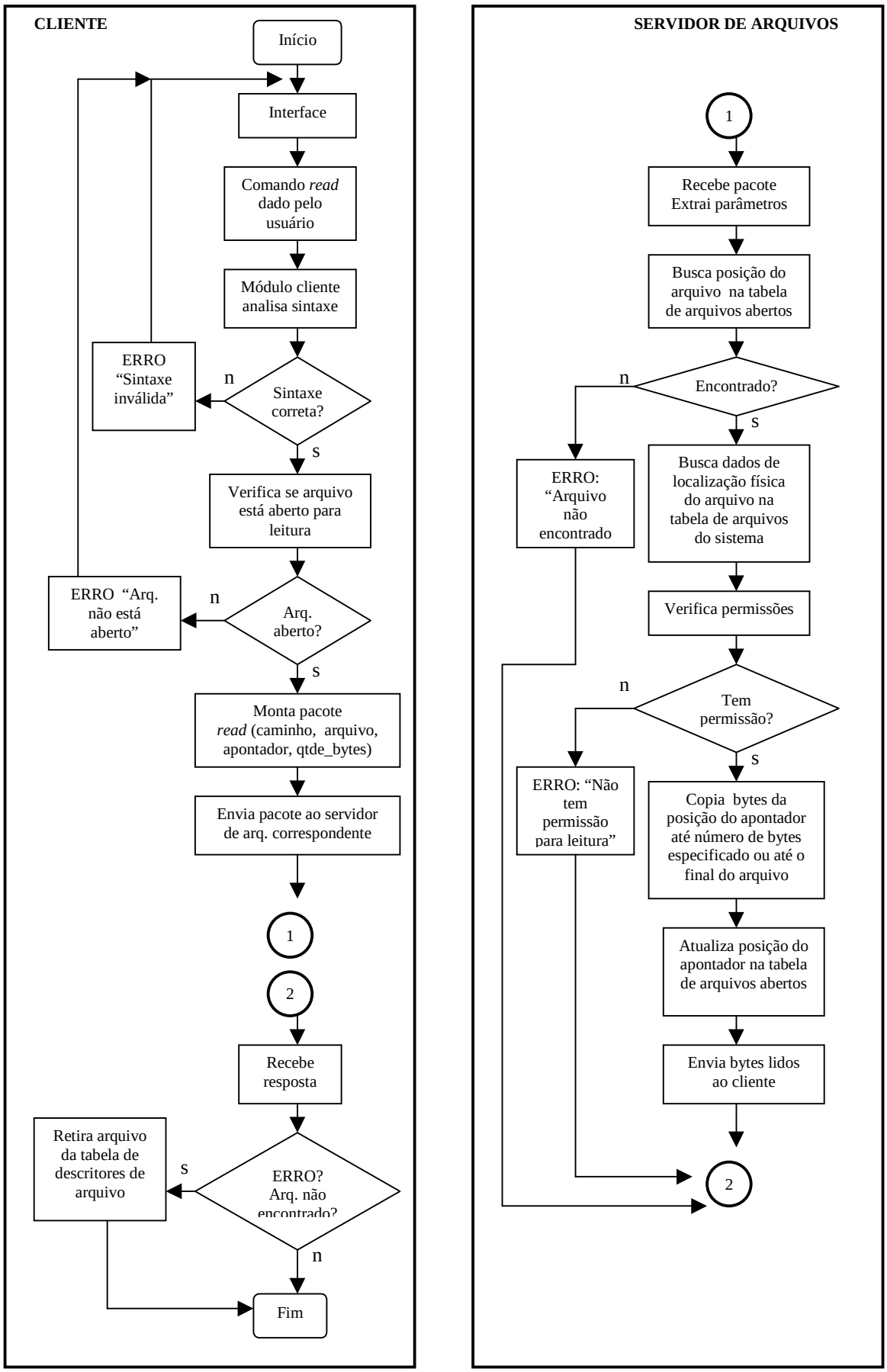
READ([NOME DE CAMINHO], ARQUIVO, QTDE_BYTES)

A primeira verificação feita para validar a leitura de um arquivo é se está aberto e presente na tabela de descritores de arquivo do cliente que solicitou a operação. A busca é feita diretamente no servidor de arquivos correspondente, já que toda a procura pelo caminho foi feita na abertura do arquivo.

Em seguida, as informações sobre o arquivo são analisadas, para saber em que servidor de arquivos se encontra. Após identificado, uma requisição ao servidor específico é feita para leitura do arquivo, enviando a posição atual do ponteiro e a quantidade de bytes a serem lidos. Como os dados são armazenados no arquivo como uma seqüência de bytes, a posição do ponteiro e a quantidade de bytes são suficientes para recuperar os dados desejados. Se a quantidade de bytes informados for maior que o existente no arquivo (a partir de onde está posicionado o apontador), o sistema mostra os dados lidos e informa a quantidade de bytes realmente lida.

Realizada a leitura, a posição do ponteiro do arquivo é modificada. A fig. 5.14 mostra a especificação da primitiva *read*.

FIGURA 5.14 – ESPECIFICAÇÃO DA PRIMITIVA *READ*



5.4.2.4 ESCRITA EM ARQUIVO - *WRITE*

O comando ***write*** permite escrever em arquivos já existente. A escrita é sempre realizada no final do arquivo e gravada como uma seqüência de *bytes*. O quadro 5.8 mostra o formato da primitiva *write*.

QUADRO 5.8 – SINTAXE DA PRIMITIVA *WRITE*

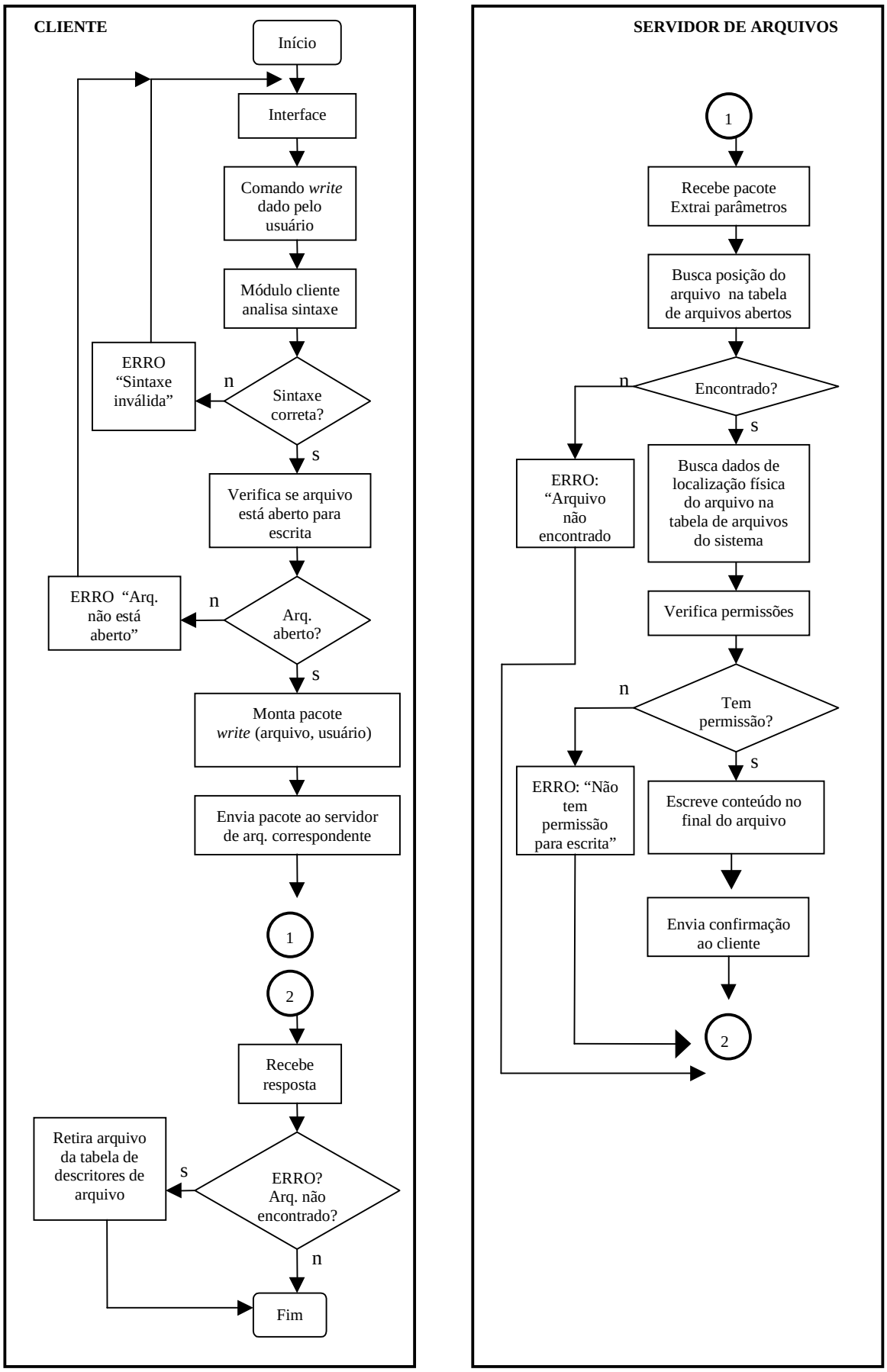
WRITE([NOME DE CAMINHO], ARQUIVO, DADOS)

Assim como acontece na leitura, para escrever dados em um arquivo este deve ter sido primeiramente aberto, e o modo leitura/escrita (*rw*) especificado na abertura. A primeira verificação feita para validar a leitura de um arquivo é se está aberto e presente na tabela de descritores de arquivo do cliente que solicitou a operação. A busca é feita diretamente no servidor de arquivos correspondente, já que a tabela de descritores de arquivo armazena a localização do arquivo quando ele é aberto.

Em seguida, as informações sobre o arquivo são analisadas, para saber em que servidor de arquivos se encontra. Após identificado, uma requisição ao servidor específico é feita para escrita no arquivo, assim como uma verificação para saber se o arquivo está bloqueado.

A fig. 5.15 mostra a especificação da primitiva *write*.

FIGURA 5.15 – ESPECIFICAÇÃO DO COMANDO *WRITE*



5.4.2.5 REMOÇÃO DE ARQUIVO - *DELETE*

O comando ***delete*** remove um arquivo fisicamente e das estruturas de controle dos servidores de arquivo e diretório. O quadro 5.9 mostra o formato da primitiva *write*.

QUADRO 5.9 – SINTAXE DA PRIMITIVA *DELETE*

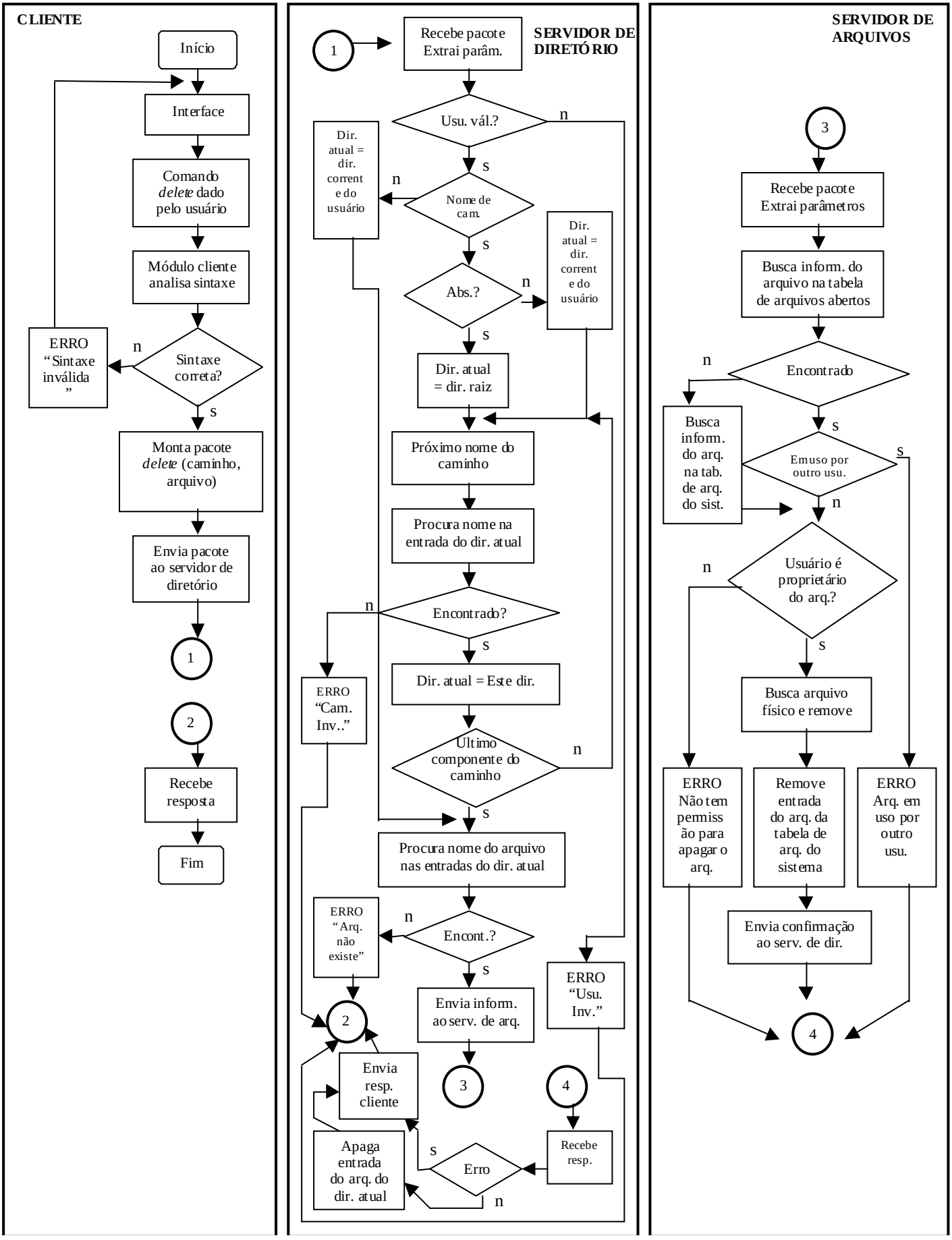
DELETE([NOME DE CAMINHO], ARQUIVO)

Para excluir um arquivo, este deve estar fechado, por isso, a remoção consiste em o cliente requisitar a exclusão ao servidor de diretórios, que faz todas as verificações e validações em nome de caminho absoluto ou relativo, conforme descrito em operações anteriores. Depois de verificar, faz contato com o servidor de arquivos correspondente.

O servidor de arquivos, por sua vez, verifica se ele não está bloqueado ou se está em uso por outro usuário. Se estiver, envia mensagem de erro ao servidor de diretórios. Se a operação puder ser realizada, o arquivo é excluído do disco e da tabela de arquivos do sistema. Neste caso, uma confirmação é enviada ao servidor de diretórios para que o arquivo seja também retirado das estruturas de diretório do cliente. O cliente recebe confirmação de exclusão ou mensagem de erro.

A fig. 5.16 mostra a especificação da primitiva *delete*.

FIGURA 5.16 – ESPECIFICAÇÃO DA PRIMITIVA *DELETE*



5.4.2.6 FECHAMENTO DE ARQUIVO - *CLOSE*

O comando ***close*** fecha o arquivo especificado, retirando-o da tabela de descritores de arquivos do cliente que solicitou a operação. Se este arquivo estava aberto e bloqueado, é então liberado aos demais usuários. O quadro 5.10 mostra o formato da primitiva *close*.

QUADRO 5.10 – SINTAXE DA PRIMITIVA *CLOSE*

CLOSE([ARQUIVO)

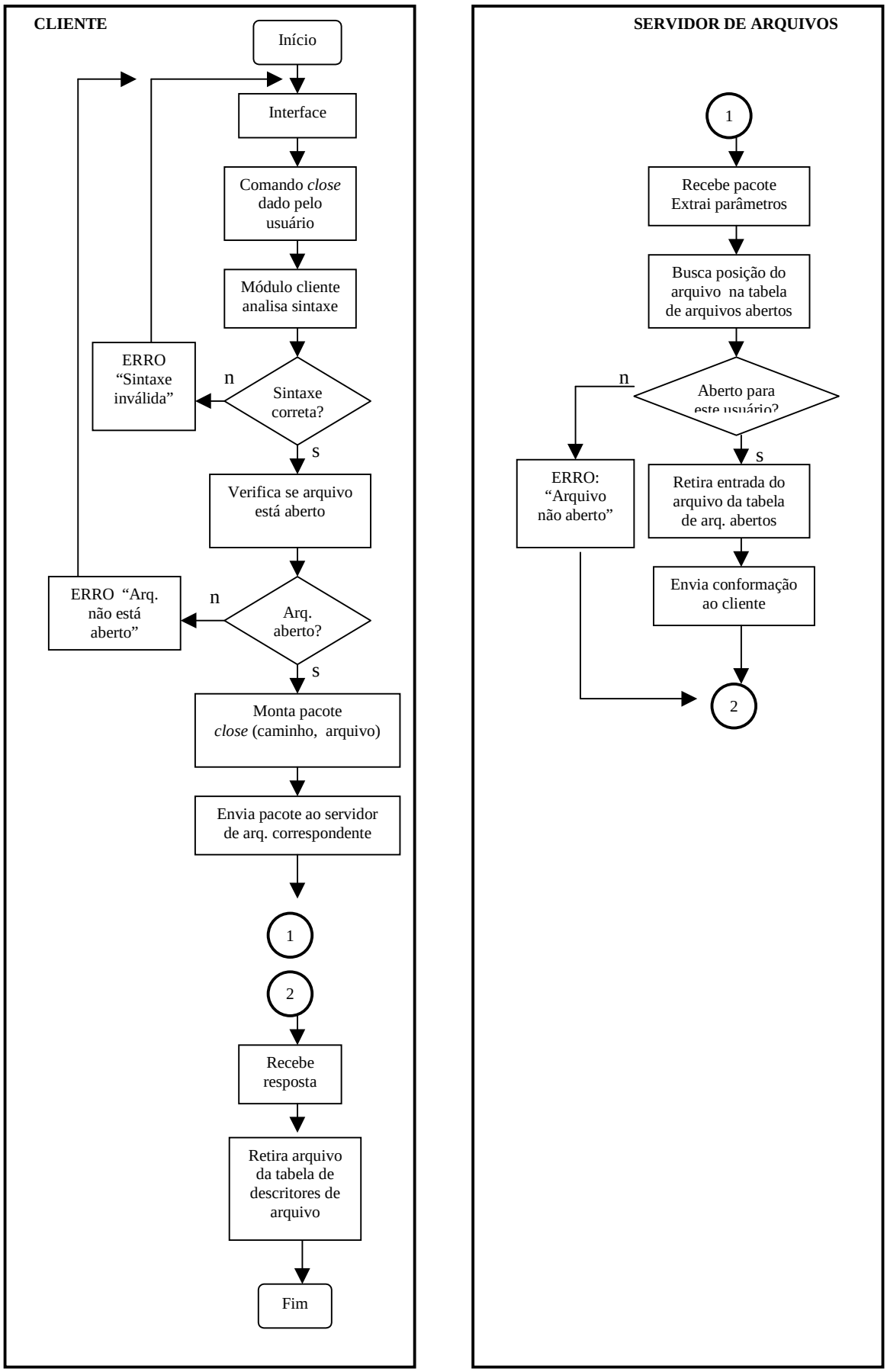
Ao ser especificado um comando *close*, uma verificação para saber se o arquivo realmente estava aberto é feita. Neste caso, informações de localização do arquivo devem ser retiradas da tabela de descritores de arquivos e da tabela de arquivos abertos do sistema.

O servidor de arquivos que recebeu a requisição retira então a entrada do arquivo da tabela de arquivos abertos do sistema, enviando confirmação ao cliente.

Ao receber a confirmação de operação bem sucedida, o cliente retira a entrada do arquivo da sua tabela de descritores de arquivos. Qualquer operação subsequente de leitura ou escrita neste arquivo exige primeiramente a sua abertura.

A fig. 5.17 mostra a especificação da primitiva *close*.

FIGURA 5.16 – ESPECIFICAÇÃO DA PRIMITIVA *CLOSE*



5.5 IMPLEMENTAÇÃO DO PROTÓTIPO

Para o desenvolvimento do protótipo de gerenciador de arquivos distribuído foi utilizada a linguagem de programação Java, mais precisamente a ferramenta de desenvolvimento JDK (*Java Developers Kit*).

5.5.1 APLICAÇÕES DISTRIBUÍDAS EM JAVA

Java traz como vantagens a independência de plataforma, fortes recursos de rede, benefícios da orientação a objetos e múltiplas linhas de execução (*Threads*) [THO97], características que facilitaram o desenvolvimento do protótipo.

A linguagem Java foi originalmente desenvolvida para eletrodomésticos, por isso, ela é simples e portátil. Ela nasceu considerando a Internet como seu ambiente de atuação. Java herdou várias características de outras linguagens de programação como Objective-C, Smalltalk, Modula 3, entre outras, com o intuito de ser uma linguagem moderna e de fácil manuseio [NEW97].

Ao contrário das ferramentas tradicionais de programação, um programa Java é compilado para o *bytecode* da *Java Virtual Machine*, que nada mais é do que uma máquina virtual idealizada pelos criadores da linguagem, e depois executada por uma implementação (em *hardware* ou *software*) desta máquina.

O *bytecode* gerado na compilação pode ser transportável para diversas plataformas como Mac/Os, Windows 95/98/NT®, Unix, Linux, por exemplo. Essa característica destaca-se neste meio por não aparecer em outras linguagens de programação sem que haja uma recompilação do código fonte original com algumas eventuais mudanças.

Além da portabilidade, a linguagem Java também se destaca pela sua confiabilidade. Ela evita erros comuns de programadores gerenciando a memória, evitando perda de memória desnecessária através da eliminação do uso de ponteiros, não deixando com que o programador tenha acesso à memória que não pertence ao seu programa [NEW97] e [LAL97].

5.5.2 COMUNICAÇÃO ENTRE PROCESSOS REMOTOS: SOCKETS

Uma das vantagens para utilização de Java para a Internet é que ela oferece facilidades para o desenvolvimento de aplicações com protocolo TCP/IP (*sockets*, datagramas) utilizado na arquitetura da Internet.

Socket é uma abstração de software usada para representar os "terminais" de uma conexão entre duas máquinas. Em cada conexão há um *socket* para cada máquina, que se comunicam através de uma porta, definida na criação do socket. Para implementações que utilizam *sockets*, é necessário estabelecer um protocolo, a ser utilizado na comunicação entre as máquinas.

5.6 ALGORITMOS DOS SERVIDORES

Para demonstrar a implementação do servidor de diretórios e do servidor de arquivos, uma listagem dos algoritmos podem ser visualizadas nas figuras 5.18 e 5.19, respectivamente. Mais detalhes da implementação estão destacados nos anexos, que contém partes do código fonte do protótipo.

FIGURA 5.18 – ALGORITMO DO SERVIDOR DE DIRETÓRIOS

```
Início Servidor de Diretórios
    loop para sempre
       Espera contato do cliente;
       Cria Thread para atendê-lo;
    fim loop
fim

Início Thread
     loop para sempre
         escolha chamada
            Conectar: valida cliente e coloca na tabela de usuários ativos;
            Mkdir: chama rotina para criar diretório;
            Chdir: chama rotina para modificar diretório corrente;
            Rmdir: chama rotina para excluir diretório, retira entrada do diretório;
            List: chama rotina para listar conteúdo do diretório;
            Open: chama rotina para abrir arquivo. Contacta com serv.de arq.;
            Delete: chama rotina de exclusão, contacta serv. arq., retira entrada;
            Create: chama rotina de criação, contacta serv. Arq., insere entrada;
            Desconectar: retira usuário da tabela de usuários ativos;
         fim escolha
     fim loop
Fim
```

FIGURA 5.19 – ALGORITMO DO SERVIDOR DE ARQUIVOS

```
Início Servidor de Arquivos
    loop para sempre
       Espera contato do cliente/servidor de diretórios;
       Cria Thread para atendê-lo;
    fim loop
fim


Início Thread
     loop para sempre
         escolha chamada
           Create: chama rotina para criar arquivo. Retorna resposta para serv. dir.;
           Open: chama rotina para abrir arquivo, insere na tab. arq.abertos, retorna resp.
                    serv. diretórios;
           Write: chama rotina para escrever em um arquivo;
           Read: chama rotina para ler arquivo, atualiza apontador, retorna conteúdo ao
                    cliente;
           Close: chama rotina para fechar arquivo. Retira da tab. arq. abertos;
           Delete: chama rotina para excluir arquivo. Retira arq. da tab. arq. Sistema,
                    retorna resposta ao serv. diretórios;
         fim escolha
     fim loop
Fim
```

6 FUNCIONAMENTO DO PROTÓTIPO

O protótipo de gerenciador de arquivos distribuído possui três módulos: um servidor de diretórios, um servidor de arquivos e um módulo cliente. O servidor de diretórios deve estar fixo em uma das máquinas da rede, enquanto o servidor de arquivos e o cliente podem estar em uma ou mais máquinas. A primeira coisa a fazer é configurar um arquivo denominado *host*, que identifica quais as máquinas serão servidoras. Este arquivo é utilizado pelos clientes e pelo servidor de diretórios.

Para demonstrar o funcionamento dos módulos, criou-se uma aplicação que permite que o usuário entre com comandos de manipulação de arquivos e diretórios. Os comandos da aplicação acionam as primitivas de comandos do gerenciador de arquivos, que executam a tarefa solicitada.

A aplicação disponibiliza os comandos através de uma janela, que contém uma barra de menus com as opções disponíveis. Para utilizar os serviços, o usuário deve conectar-se ao sistema e fornecer seu *login* e senha (fig. 6.1).

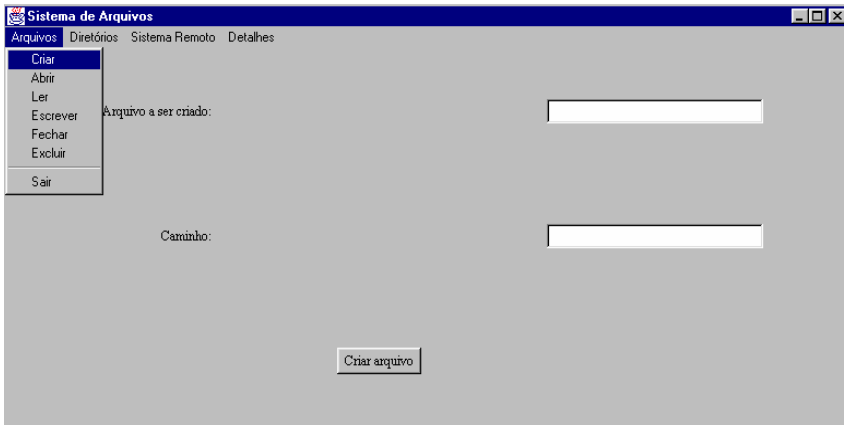
FIGURA 6.1 – TELA DE CONEXÃO DA APLICAÇÃO

Após receber uma resposta de validação do *login* e senha, o usuário poderá utilizar os serviços disponibilizados pela aplicação.

Em cada solicitação feita pelo usuário, a aplicação apresenta os parâmetros necessários que devem ser informados. Por exemplo, se for executado um comando para criar um arquivo,

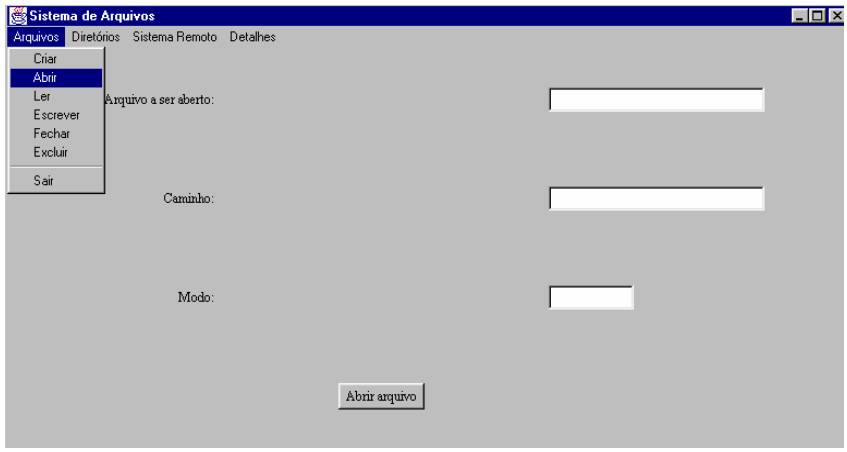
será requisitado o nome do arquivo a ser criado e, opcionalmente, o caminho de diretórios onde o arquivo deve ser criado (fig. 6.2).

FIGURA 6.2 – CRIAÇÃO DE UM ARQUIVO



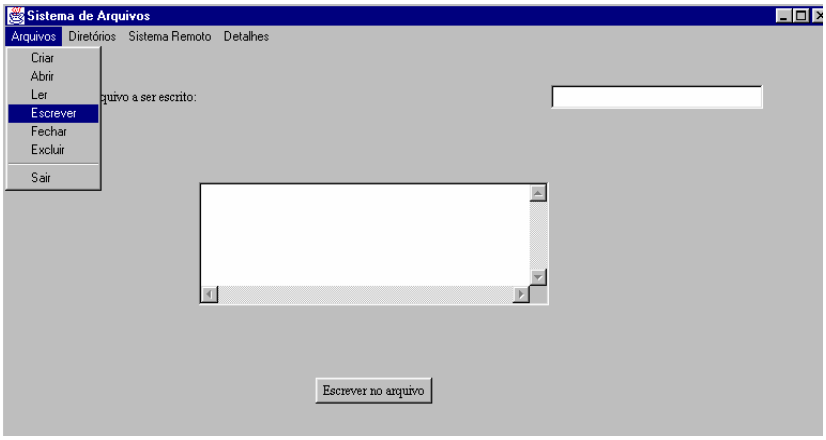
Na abertura de um arquivo, é requisitado o nome do arquivo e opcionalmente o caminho onde se encontra e o modo de abertura (*r* para somente leitura e *rw* para leitura e escrita), conforme mostra fig. 6.3.

FIGURA 6.3 – ABERTURA DE UM ARQUIVO



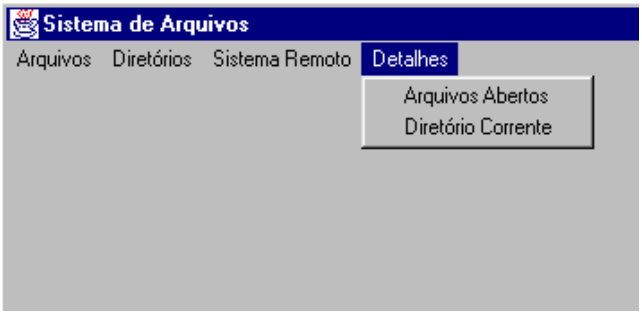
Para escrever em um arquivo, este deve ser previamente aberto, e o nome do arquivo e o conteúdo a ser escrito deve ser informado (fig. 6.4). E assim segue para os demais comandos.

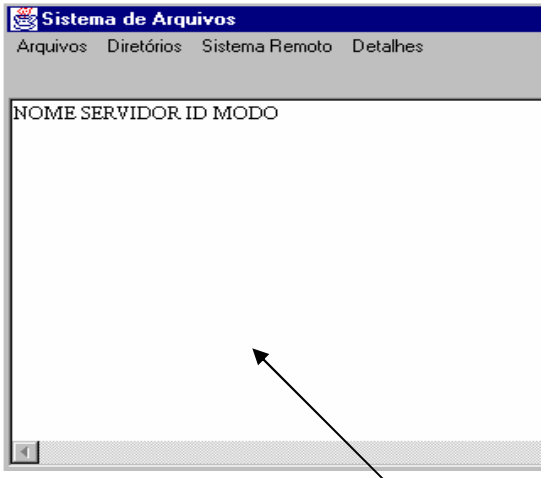
FIGURA 6.4 – ESCRITA EM UM ARQUIVO

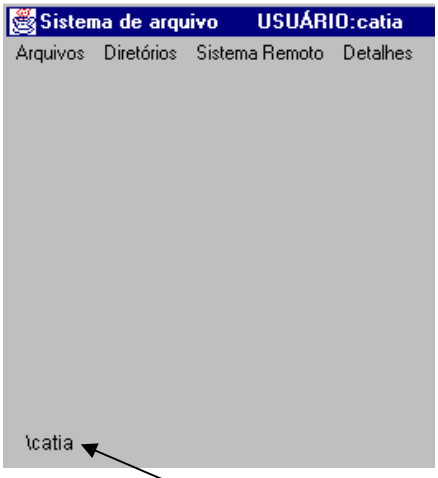


Um menu denominado *detalhes* permite visualizar o diretório corrente e quais os arquivos abertos (fig. 6.5).

FIGURA 6.5 – MENU DETALHES







Arquivos abertos

Diretório corrente

Em nenhum momento o cliente preocupa-se com a localização física dos arquivos e quais computadores do sistema são servidores de arquivos ou de diretórios.

6.1 RESULTADOS OBTIDOS NOS TESTES

Os testes foram realizados no Laboratório do Protem e no Laboratório do bloco G da Universidade Regional de Blumenau, sendo que foram efetuados os testes em diferentes microcomputadores para comprovar o funcionamento do protótipo.

No Laboratório do Protem, o módulo cliente foi instalado em 3 computadores: Mole, cuja identificação é mole.inf.furb.rct-sc.br, Poa, sob a identificação poa.inf.furb.rct-sc.br e Laguna, com a identificação laguna.inf.furb.rct-sc.br. O servidor de diretórios foi instalado no computador Mole (mole.inf.furb.rct-sc.br) e o servidor de arquivos neste mesmo computador e no computador Poa (poa.inf.furb.rct-sc.br).

Como comentado anteriormente, utilizou-se o recurso de *socket*, que Java oferece para a comunicação de computadores. Para isso, é necessário informar uma porta de comunicação a ser utilizada pelos módulos que irão se comunicar. Definiu-se a porta 23 para comunicação dos clientes com o servidor de diretórios e a porta 24 para a comunicação do servidor de diretórios e dos clientes com o servidor de arquivos.

Os resultados foram satisfatórios e atenderam a expectativa do objetivo do protótipo, confirmando o funcionamento do mesmo.

7 CONSIDERAÇÕES FINAIS

O desenvolvimento do protótipo de gerenciador de arquivos para ambiente distribuído trouxe uma ampla bagagem de conhecimentos na área de sistemas operacionais e ambientes distribuídos. O estudo realizado para o desenvolvimento do mesmo veio confirmar a complexidade que envolve estas áreas e a grande variedade de itens necessários para a compreensão do seu funcionamento.

7.1 CONCLUSÕES

O protótipo alcançou seus objetivos no sentido de implementar um gerenciador de arquivos que ofereça serviços de manipulação de arquivos e diretórios, de forma transparente quanto a localização física dos arquivos, que podem estar espalhados em um ambiente distribuído.

Operações de armazenamento e recuperação de informações através de arquivos são disponibilizadas, sem que o usuário tenha que se preocupar em qual das máquinas da rede seu arquivo será armazenado ou onde deverá procurar o arquivo na hora de recuperar seus dados. A organização dos diretórios em árvore, concluída como sendo a mais adequada foi implementada com sucesso e os arquivos e diretórios criados são organizados logicamente nesta hierarquia, aparecendo para o usuário como uma árvore de diretórios.

Alguns aspectos de segurança dos arquivos foram considerados e tratados através de permissões de acesso e manipulação. Não foram tratadas questões de falhas no ambiente ou erros que poderão ocorrer caso as máquinas que contém os servidores venham a falhar.

Através do bloqueio aos arquivos abertos, pode-se observar e constatar o funcionamento mínimo das semânticas de compartilhamento e a sua importância na obtenção da integridade dos dados.

O desenvolvimento do protótipo demonstrou com sucesso alguns dos aspectos a serem considerados na implementação de sistemas de arquivos e os controles necessários, principalmente quando se trata de ambiente distribuídos.

7.2 SUGESTÕES PARA TRABALHOS FUTUROS

Ao desenvolver o protótipo de gerenciador de arquivos distribuídos, apenas alguns dos conceitos estudados foram utilizados, por questões de complexidade e, consequentemente, tempo que um sistema completo exigiria.

O controle e gerenciamento dos blocos físicos do disco é um dos complementos que poderiam ser implementados. Como sugestão observada na bibliografia, um controle dos blocos livres e ocupados do disco, através de listas e sua ocupação adequada seria uma forma de prover este gerenciamento.

Mecanismos de detecção de falhas no ambiente também poderiam ser implementados, para garantir que dados não se percam durante uma sessão mal finalizada.

Restrições quanto ao uso dos diretórios, através de permissões, de forma similar à realizados nos arquivos podem garantir uma maior segurança no acesso às informações.

REFERÊNCIAS BIBLIOGRÁFICAS

[COM88] COMER, Douglas; FOSSUM, Timothy. **Operating system design**. Rio de Janeiro : Prentice-Hall do Brasil.

[DAM96] DAMASCENO JUNIOR, Peter. **Aprendendo Java :** programação na Internet. São Paulo : Erica, 1996.

[KIR88] KIRNER, Cláudio. **Sistemas operacionais distribuídos** : aspectos gerais e análise de sua estrutura. Rio de Janeiro : Campus, 1988.

[LAL97] LALANI, Suleiman. **Java :** biblioteca do programador. São Paulo : Makron Books, 1997.

[MAC97] MACHADO, Berenger; MAIA, Luiz Paulo. **Arquitetura de sistemas operacionais**. Rio de Janeiro : Livros Técnicos e Científicos Editora, 1997.

[NEW97] NEWMAN, Alexander. **Usando Java**. Rio de Janeiro : Campus, 1997.

[SIL94] SILBERSCHATZ, Abraham; GALVIN, Peter B. **Operating system concepts**. Reading, Massachusetts : Addison-Wesley, 1994.

[STR84] STRACK, Jair. **Sistemas de processamento distribuído**. Rio de Janeiro : Livros Técnicos e Científicos Editora, 1984.

[TAN87] TANENBAUM, Andrew S. **Operating systems** : design and implementation. Englewood Cliffs : Prentice-Hall, 1987.

[TAN95] TANENBAUM, Andrew S. **Sistemas operacionais modernos**. Rio de Janeiro : Prentice-Hall do Brasil, 1995.

[THO97] THOMAS, Michael D.; PATEL, Patrick R.; HUDSON, Alan D.; BALL JR, Donald A. **Pogramando em Java para a Internet** : um guia para criar aplicações dinâmicas e interativas. São Paulo : Makron Books, 1997.

[VAH96] VAHALIA, Uresh. **Unix internals** : The new frontiers. New Jersey : Prentice Hall, 1996.