UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS DE SÃO CARLOS
Área Ciências de Computação e Matemática Computacional

**Utilização de Disco Ótico (WORM) como
Medida de Segurança de Arquivos em Redes
Locais de Computadores**

Luiz A. F. Bezerra

Orientadora: Prof^a Dr^a Regina H. C. Santana

**Dissertação apresentada ao Instituto de Ciências Matemáticas de São Carlos, da Universidade de
São Paulo, como parte dos requisitos para obtenção do título de Mestre em Ciências - Área:
Ciências de Computação e Matemática Computacional.**

**São Carlos
1993**

DEDICATÓRIA

*Dedico este trabalho
À minha esposa Regina;
aos meus filhos:
Patricia, Rodrigo e Rafael;
em memória de meu pai e avós;
à minha mãe e aos meus irmãos.*

AGRADECIMENTOS

À todas as pessoas que de alguma forma contribuíram para a realização deste trabalho, expresso meus agradecimentos.

À minha orientadora Prof^a Dr^a Regina H. C. Santana, pelo mérito de sua orientação, ensinamento, dedicação e tolerância.

Ao Prof Dr Marcos J. Santana pelas valiosas sugestões que contribuíram para a definição deste trabalho.

Ao Prof Dr Francisco A. M. Duarte e Prof Dr Raysildo B. Lôbo, pelo incentivo e apoio oferecido durante todo o programa de mestrado.

Ao Departamento de Genética e Matemática Aplicada à Biologia da Faculdade de Medicina de Ribeirão Preto - USP, na pessoa do, Prof Dr Moacyr A. Mestriner, pela oportunidade e condições proporcionadas.

Ao Instituto de Ciências Matemáticas de São Carlos - USP, pelo ambiente de trabalho e pela fonte de conhecimentos de seus professores.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), pelo apoio financeiro dado ao grupo de Sistemas Distribuídos e Programação Concorrente.

RESUMO

Esta dissertação apresenta um sistema auxiliar de arquivos baseado em disco WORM, desenvolvido com o propósito de proporcionar aos servidores de arquivos de um ambiente computacional distribuído, procedimentos para transferência de arquivos originalmente armazenados em disco magnético, para disco WORM. O mecanismo proposto é transparente ao usuário e tem como objetivos aumentar tanto a segurança quanto a disponibilidade de espaço no servidor de arquivos. Estes objetivos são atingidos através da replicação de arquivos importantes e da transferência de arquivos de baixa probabilidade de acesso.

Considerando-se os aspectos de segurança e de disponibilidade de espaço, três tipos de arquivos para gravação em WORM são propostos: crítico (quando o usuário necessita da disponibilidade do arquivo mesmo em presença de falhas), normal (qualquer arquivo regular do sistema) e morto (alguns arquivos eliminados do sistema).

O sistema auxiliar de arquivos WORM foi implementado utilizando-se recursos da linguagem de programação C e do sistema operacional MINIX, simulando-se o disco ótico WORM através de disquete.

ABSTRACT

This dissertation discusses a WORM-based auxiliar file system, developed to improve LAN-based file servers by means of special file-transfer procedures, allowing files originally stored on Winchester disks to be moved to WORM disks. The transfer-mechanism proposed is user-transparent and aims to improve both the file-server security and disk space availability.

Security is improved by means of replication of important files on the WORM system; disk-space usage is optimized by moving files with low-access probability to the WORM system.

Three types of files are proposed to implement the transfer-mechanism: special files (available even when failures occur); ordinary files (any regular file of the system) and unused files (files that have been removed from the system).

The auxiliar file system was implemented in a MINIX-based environment (using the C programming language) and emulates the WORM disk on floppy-disk.

SUMÁRIO

1 INTRODUÇÃO	1
2 SISTEMAS DISTRIBUÍDOS	4
2.1 INTRODUÇÃO	4
2.2 MODELOS DE SISTEMAS DISTRIBUÍDOS	5
2.3 REDES DE COMPUTADORES	8
2.4 TÓPICOS DE PROJETO DE SISTEMA DISTRIBUÍDO	15
2.5 COMENTÁRIOS	19
3 SERVIDORES DE ARQUIVOS	21
3.1 INTRODUÇÃO	21
3.2 CONCEITOS	24
3.3 ORGANIZAÇÃO	25
3.4 ESTRUTURA LÓGICA E FÍSICA	26
3.5 EXEMPLOS	29
3.6 TÓPICOS DE PROJETO DE SERVIDOR DE ARQUIVOS	31
3.7 CONTROLE DE ACESSO	33
3.8 CONFIABILIDADE	34
3.9 COMENTÁRIOS	36
4 SISTEMAS DE ARQUIVOS BASEADOS EM DISCOS ÓTICOS WORM	37
4.1 INTRODUÇÃO	37
4.2 DISCOS ÓTICOS: CARACTERÍSTICAS FÍSICAS	40
4.3 DISCOS ÓTICOS: CARACTERÍSTICAS LÓGICAS	43

4.4 EXEMPLOS DE SISTEMAS DE ARQUIVOS UTILIZANDO DISCOS ÓTICOS	44
4.5 COMENTÁRIOS	46
5 PROPOSTA DE UM SISTEMA AUXILIAR DE SEGURANÇA DE ARQUIVOS	48
5.1 INTRODUÇÃO	48
5.2 IMPORTÂNCIA DO SISTEMA AUXILIAR DE SEGURANÇA DE ARQUIVOS	49
5.3 SISTEMA OPERACIONAL MINIX	53
5.4 SERVIDORES DE ARQUIVAMENTO (ARCHIVE SERVERS)	57
5.5 CARACTERIZAÇÃO DO SISTEMA AUXILIAR DE ARQUIVOS WORM	60
5.6 COMENTÁRIOS	67
6 IMPLEMENTAÇÃO DO SISTEMA AUXILIAR DE ARQUIVOS WORM	70
6.1 INTRODUÇÃO	70
6.2 MÓDULOS COMPONENTES DO SISTEMA AUXILIAR DE ARQUIVOS WORM	71
6.3 ESTRUTURA INICIAL	74
6.4 PROCEDIMENTOS TRANSPARENTES AO USUÁRIO	75
6.5 COMANDOS DISPONÍVEIS AO USUÁRIO	77
6.6 OUTROS COMANDOS DE APOIO AO USUÁRIO	78
6.7 TÓPICOS DE DESTAQUE DA IMPLEMENTAÇÃO	81
6.8 TESTES	83
6.9 COMENTÁRIOS	85
7 COMENTÁRIOS FINAIS	89
7.1 REVISÃO BIBLIOGRÁFICA	89
7.2 SISTEMA AUXILIAR DE ARQUIVOS WORM	91
7.3 CONTRIBUIÇÃO DO TRABALHO PROPOSTO	95
7.4 CONTINUIDADE DO TRABALHO	96
BIBLIOGRAFIA	97

APÊNDICE A - Código Fonte dos processos OPEN e CLOSE	99
APÊNDICE B - Código Fonte dos procedimentos do saa-WORM	100
APÊNDICE C - Comandos do saa-WORM	119

INTRODUÇÃO

A evolução dos sistemas de computação tem sido marcante para o homem, graças a eletrônica. Inicialmente, os sistemas propostos atendiam a um único usuário por vez - sistemas monousuários, que apesar dos poucos recursos, foram revolucionários para a época. A intensificação de uso desses computadores permitiram a criação de sistemas mais sofisticados, os sistemas multi-usuários como os Sistemas de Tempo Compartilhado, e os Sistemas em Tempo Real. Do aperfeiçoamento desses sistemas surgiram os sistemas atuais: os sistemas distribuídos e os sistemas de rede local de computadores. Em particular, sistemas distribuídos baseados em redes locais, oferecem ao usuário um grande potencial de computação, porque colocam a sua disposição recursos transparentes e compartilhados por vários usuários. Esses recursos podem ser diferentes tipos de servidores que são acessados através de uma rede de computadores.

Os Sistemas Distribuídos apresentam como característica principal a transparência, isto é, são vistos pelos seus usuários como sistemas centralizados apesar de utilizarem múltiplas e independentes unidades de processamento [TA85].

Os Sistemas Distribuídos baseados em Redes Locais de Computadores estão substituindo em larga escala os grandes sistemas tradicionais centralizados porque permitem descentralizar o processamento de informações e compartilhar recursos caros ou pouco utilizados por todos os componentes da rede. Como vantagem se tem menor custo e maior segurança uma vez que o sistema dispõe de varias unidades de processamento, não ficando dependente de uma unidade central de processamento como nos sistemas tradicionais.

Pelo fato dos sistemas computacionais distribuídos servirem a muitos usuários, eles apresentam alguns pontos críticos tais como o desempenho, a segurança e disponibilidade das informações. Neste sentido, os servidores de arquivos são provavelmente o ponto mais crítico do sistema. Para administrar estes sistemas são utilizados mecanismos capazes de assegurar a integridade e disponibilidade das informações [SV84].

Considerando-se a dificuldade de administrar a segurança e a disponibilidade das informações nos sistemas computacionais distribuídos baseados em rede local de computadores, julgou-se relevante propor um sistema auxiliar de arquivos baseado em disco ótico de única escrita (WORM).

Devido às características e ao advento dos discos óticos em particular do tipo WORM [GA91] julgou-se ser este dispositivo adequado ao que se propõe neste trabalho. A utilização dos discos óticos em relação a outros dispositivos, tais como: discos e fitas magnéticas, apresentam diversas vantagens:

- não se alteram em presença de campos magnéticos,
- grande capacidade de armazenamento,
- baixo custo por byte armazenado,
- gravação única, i.e., não permitem que seu conteúdo seja alterado após ter sido gravado.

O propósito do sistema auxiliar de arquivos baseado em disco WORM, é de realizar para os servidores de arquivos os procedimentos de transferência de arquivos [SA81] dos discos magnéticos para o WORM automaticamente. Esta transferência se faz necessário por questões de segurança e de disponibilidade de espaço ao servidor de arquivos. Com relação a segurança o sistema auxiliar de arquivos baseado em disco WORM cuida para que determinados arquivos sejam copiados para o disco WORM, automaticamente e o mais breve possível, toda vez que forem alterados. Com o propósito de aumentar a disponibilidade de espaço do servidor de arquivos, o sistema auxiliar de arquivos baseado em disco WORM, dispõe de procedimentos que monitoram todos os arquivos do sistema de arquivos, agindo em função da data do último acesso do arquivo [SA87] [SA81]. É possível também a participação de usuários privilegiados que podem solicitar ao sistema auxiliar de arquivos, a cópia de um arquivo qualquer para o disco WORM. A recuperação (ou re-instalação) dos arquivos do disco WORM para o disco magnético pode ser feita também de maneira transparente para o usuário ou através de solicitação do próprio usuário.

O enfoque principal do trabalho proposto está nas implicações geradas por falha irrecuperável de um dispositivo de armazenagem de informação, sendo de menor interesse abordar outras alternativas de perda de integridade e disponibilidade de informações causadas por violação ou uso indevido dos usuários do sistema, tais como: vírus, "hackers", ou mesmo falhas do próprio usuário. No entanto, a característica de única escrita do disco WORM, transforma-o em um banco de versões de arquivos sendo possível em alguns casos a recuperação de uma cópia de arquivo removido indevidamente pelo próprio usuário ou alterado de maneira fraudulenta.

Os capítulos seguintes visam apresentar o suporte básico para o desenvolvimento do trabalho e a definição, implementação e teste do sistema auxiliar de arquivos baseado em disco ótico WORM.

O Capítulo dois apresenta as características relevantes de sistemas distribuídos e de rede local de computadores.

No Capítulo três são abordados aspectos gerais de servidores de arquivos, dando maior ênfase a segurança e disponibilidade das informações.

No Capítulo quatro são apresentados sistemas de arquivos baseados em disco ótico WORM discutindo-se características físicas, lógicas e exemplos.

O Capítulo cinco trata da definição proposta para o sistema auxiliar de arquivos baseado em disco WORM. Neste capítulo é dada ênfase à necessidade dos servidores de arquivos possuírem mecanismos não só na área de segurança como também no que diz respeito ao gerenciamento de espaço disponível com atenção especial aos arquivos com baixa probabilidade de acesso.

No Capítulo seis são discutidos os principais tópicos da implementação do sistema auxiliar de arquivos WORM. Apesar da implementação ter sido feita para uma unidade de disquete simulando o disco WORM, todos os mecanismos definidos para o sistema foram explorados.

No Capítulo sete são apresentados os comentários finais sobre o sistema auxiliar de arquivos baseado em disco WORM, sua contribuição para os sistemas atuais e para os sistemas futuros.

SISTEMAS DISTRIBUÍDOS

2.1. INTRODUÇÃO

No início da década de 60, linhas telefônicas dedicadas ligavam alguns computadores a impressoras remotas e entrada de dados em diferentes localidades [WI91]. Com a decorrente evolução tecnológica e a intensificação dos estudos de redes, houve redução nos custos de comunicação e de hardware, possibilitando o desenvolvimento das redes de comunicação de computadores [KI86]. As redes de comunicação de computadores permitem que os computadores troquem dados de forma rápida e confiável, através de linhas de comunicação e protocolos de software.

Os Sistemas Distribuídos são caracterizados por terem vários computadores interconectados em rede, que têm como propósito o processamento cooperativo de um grande problema ou o processamento simultâneo de várias tarefas individuais porém, relacionadas. Apesar de todos os computadores integrantes do sistema distribuído terem uma memória local e dispositivos de entrada e saída, operacionalmente aparentam ser, ao usuário, um único computador [WI91]. Esta característica evidencia o conceito básico de sistema distribuído: a transparência [TA85].

Segundo Mullender [MU89], não basta a transparência para definir um sistema distribuído. Ele tem que ser tolerante a falhas, isto é, qualquer parte simples com falha não pode parar todo o sistema. Visto que um sistema distribuído é constituído de múltiplas partes de hardware e de software, ele pode ser mais confiável que os sistemas centralizados porque tem suas funções repetidas. Se um processador falha outro pode tomar o seu lugar. Seus arquivos também podem ser armazenados em várias unidades de discos, assim, se uma unidade de disco falhar, nenhuma informação será destruída.

A definição de sistema distribuído é baseada em várias características, tais como: multiplicidade de componentes similares, interconexão de elementos de processamento, transparência, ausência de estruturas de controle hierárquico, tolerância a falhas, etc. Dependendo da característica enfocada pode ser difícil a distinção entre sistemas distribuídos e sistemas centralizados.

2.2 MODELOS DE SISTEMAS DISTRIBUÍDOS

Considerando-se características de hardware tais como: interconexão dos computadores, compartilhamento de memória e periféricos e características de software do sistema operacional, é possível definir-se dois modelos de sistemas distribuídos: os fracamente acoplados e os fortemente acoplados

2.2.1 Sistemas Fracamente Acoplados

Sistemas fracamente acoplados são caracterizados pelo uso eficiente da potência e flexibilidade das estações de trabalho e computadores mono-usuários, permitindo a eles compartilharem dados e recursos localizados em outros computadores servidores via rede local de alta velocidade Figura 2.1 [CO88].

Fig.2.1 - Sistema Fracamente Acoplado

2.2.2 Sistemas Fortemente Acoplados

Sistemas fortemente acoplados são caracterizados pelos sistemas de computadores que possuem dois ou mais processadores com capacidades semelhantes de processamento sob controle de um único sistema operacional integrado e que compartilham uma memória comum, canais de entrada e saída e periféricos

Fig.2.2 - Sistema Fortemente Acoplado

Dentre os dois modelos apresentados, pretende-se utilizar o sistema fracamente acoplado por apresentar muitos aspectos importantes. Coulouris & Dollimore [CO88], identificam alguns fatores e tendências que ressaltam a importância do surgimento dos sistemas distribuídos fracamente acoplados:

- O custo do processador VLSI e componentes de memória têm diminuído drasticamente,
- Tecnologias de redes de computadores de alta velocidade estão amplamente disponíveis a um custo moderado,
- O serviço interativo obtido dos computadores centralizados é geralmente de baixa qualidade com longo e imprevisível tempo de resposta, interfaces restritas de usuário e dificuldade de configuração de hardware e software para atender aos usuários,
- Aumento de facilidades de aplicações diversas são requeridas pelos usuários, como exemplo: realizar comunicação baseada em rede de computadores, recuperação complexa de informação, interfaces de usuário gráficas e altamente interativa.

A importância dos sistemas distribuídos não é só econômica, eles são necessários, pois as interfaces homem-máquina têm se tornado mais interativas, exigindo respostas cada vez mais rápidas e complexas (imagem e som). Além destes aspectos, existem outros igualmente importantes. Dentre eles, cabe citar: [MU89]

- *Modularidade*: Interfaces entre partes do sistema devem ser projetadas com maior cuidado que em sistemas centralizados. Como consequência, os sistemas distribuídos são construídos de maneira mais modular do que os sistemas centralizados. A interface entre módulos são geralmente interfaces de chamada remota de procedimento, que automaticamente impõe um certo padrão para interfaces inter-módulos.
- *Expandibilidade*: Permitem o crescimento incremental. Para aumentar a capacidade de armazenagem ou de processamento, pode-se acrescentar novos módulos. Por exemplo, servidores de arquivos, servidores de impressão, estações de trabalho, etc.
- *Disponibilidade*: Replicação de dados e redundância de recursos que podem falhar fazem com que o sistema distribuído esteja disponível mesmo quando módulos não essenciais falhem.
- *Escalabilidade*: A capacidade de qualquer componente de um sistema centralizado impõe limitação na capacidade máxima do sistema. Preferencialmente, os sistemas distribuídos não têm partes centralizadoras, assim não existe a restrição de tamanho máximo do sistema. Naturalmente, existem outros fatores que restringem a expansão de um sistema, mas os projetistas dos sistemas distribuídos escolheram algoritmos que permitem a ampliação de um número muito grande de componentes.
- *Segurança*: Disponibilidade é um dos aspectos de segurança. Um sistema seguro não basta apenas ser disponível, mas responder corretamente, mesmo quando ocorrem falhas. Os algoritmos usados em um sistema distribuído devem permitir ao sistema comportar-se corretamente em qualquer circunstância em que se encontre suas partes, em funcionamento correto ou em falha.

Um sistema distribuído depende inteiramente de uma rede de computadores, com adequado desempenho e segurança, para transmissão de dados e controles entre os elementos a ela conectados. A seguir aspectos gerais sobre redes de computadores.

2.3 REDES DE COMPUTADORES

As redes de computadores foram desenvolvidas para suprir as necessidades de comunicação de dados e de controles entre elementos locais e de longa distância, com diferentes desempenhos e requisitos de segurança [CO88].

Tem-se usado o termo *rede de computadores* indistintamente para *rede de comunicação de computadores* como para a *própria rede de computadores*. Quando se considera aspectos de hardware são ambas idênticas, porém, com relação ao software, apresentam grande diferença. A rede de comunicação de computadores caracteriza-se por preservar os sistemas operacionais dos computadores hospedeiros, e por exigir do usuário o conhecimento tanto da localização dos recursos na rede quanto a forma de utilizá-los. Já a rede de computadores, caracteriza-se por apresentar um único sistema operacional integrado que gerencia todo o conjunto de recursos da rede, eliminando a necessidade do usuário de conhecer detalhes de localização e acesso aos elementos da rede. A evolução e refinamento do conceito de rede de computadores resultou o sistema distribuído [KI86].

2.3.1 Sistemas de Comunicação

O sistema de comunicação que possibilita interligar os computadores hospedeiros recebe várias denominações, tais como: subsistema de comunicação ou sub-rede. O subsistema de comunicação é fundamental na estrutura de uma rede, enquanto que os computadores hospedeiros são importantes pelos aspectos de aplicação [KI86] [ME84].

Um subsistema de comunicação é constituído pelo meio de transmissão chamado de circuito ou canal de transmissão e pelos elementos de comutação constituídos de interfaces ou processadores denominados: computador de comunicação, comutador de pacotes, nó de comunicação, etc. A Figura 2.3 ilustra uma estrutura de rede de comunicação de computadores.

Os subsistemas de comunicação utilizam para a interligação dos nós, um dos seguintes tipos de canais de transmissão: canais de conexão ponto-a-ponto e canais de difusão [KI86].

Fig.2.3 - Estrutura de Uma Rede de Comunicação.

Os canais de conexão ponto-a-ponto possibilitam a comunicação entre pares de nós da sub-rede. A comunicação entre dois nós poderá se dar indiretamente por meio de nós intermediários, caso não exista interligação direta entre eles. Quando isto acontece, a sub-rede de comunicação é também chamada de armazena e envia. A interligação entre os nós pode ser realizada por meio de linhas telefônicas, linhas dedicadas, enlaces de micro-ondas, etc.

Na sub-rede de difusão ou multiponto, um único canal de comunicação é compartilhado por todos os nós. Neste tipo de comunicação se faz necessário mecanismos de arbitragem para evitar conflitos de acesso ao meio de transmissão. O controle de acesso pode ser centralizado ou descentralizado (distribuído). No controle centralizado, um computador em especial é encarregado de informar quando cada nó pode transmitir. Enquanto que no controle distribuído, cada nó decide por conta própria quando fazer sua transmissão. As sub-redes de difusão podem ser divididas em sub-rede estática e sub-rede dinâmica. O que caracteriza a sub-rede estática é a parcela de tempo atribuída a cada nó independentemente de querer ou não transmitir. Já a sub-rede dinâmica é caracterizada pela demanda e pela participação de um árbitro central ou distribuído que indica qual dos nós poderá utilizar o canal.

2.3.2 Meios de Transmissão

Os meios de transmissão utilizados em sub-sistemas de comunicação são baseados em um meio físico que depende da dimensão geográfica e aplicação da rede de computadores. Os sistemas fortemente acoplados, multiprocessadores, utilizam fiações impressas ou fios paralelos para interligação das placas. Neste caso a distância entre os elementos é pequena e a velocidade de transmissão é muito grande. As redes de pequena e média dimensão geográfica (redes locais de computadores) utilizam par de fios trançados, cabo coaxial e fibra ótica, geralmente adequados aos sistemas distribuídos. Para as redes de longa distância os meios mais comuns são as linhas telefônicas e sistemas de micro-ondas, incluindo a utilização de satélites.

O subsistema de comunicação de pequena e média distância proporciona velocidade e características adequadas aos sistemas distribuídos, sendo oportuna uma análise geral dos seus meios de transmissão: par trançado, cabo coaxial e fibra ótica.

O par trançado é constituído por dois fios enrolados um sobre o outro em espiral, com o propósito de reduzir a interferência sobre cada fio e permitir que as propriedades elétricas do meio seja constante por toda a extensão. É utilizado em subsistemas de comunicação que exigem baixo custo e baixas velocidades, sendo possível atingir taxas de transmissão de alguns megabits por segundo. Seu alcance é de algumas centenas de metros e pode ser constituída de dezenas de elementos. Sua principal desvantagem é a suscetibilidade a ruídos e interferências que pode ser minimizada com blindagens adequadas [ME84] [KI86].

O cabo coaxial é formado por dois condutores, um circundando o outro, separados por um dielétrico. O cabo coaxial pode ser empregado na transmissão banda base e banda larga. A transmissão em banda base se dá por meio de um único canal de dados. Já a banda larga trabalha com múltiplos canais de transmissão simultânea de dados, derivados da multiplexação por divisão de frequência [ME84] [KI86].

A transmissão banda base alcança 10 megabits por segundo entre centenas de elementos e em distâncias menores que 1 quilômetro. Sua principal desvantagem é a atenuação dos sinais, principalmente em altas frequências.

A transmissão banda larga atinge até 20 megabits por segundo entre centenas ou milhares de elementos e em distâncias de dezenas de quilômetros. A sua desvantagem está na complexidade de instalação e manutenção.

O cabo de fibra ótica é formado por um conjunto de minúsculos tubos de fibra de vidro que transmitem sinal de luz codificado na frequência do infravermelho (10^{14} a 10^{15} Hz), emitidos por diodos emissores de luz ou lasers de semicondutores. A fibra ótica apresenta a melhor taxa de transmissão comparativamente aos outros meios, alcançando atualmente 1000 megabits por segundo e em distâncias de 1 quilômetro [TA89]. Além da alta taxa de transmissão as fibras óticas apresentam alta imunidade a ruídos. Sua desvantagem se deve ao custo relativamente alto e a dificuldade de implementação de canais de difusão multipontos [ME84] [KI86].

2.3.3 Topologias

A topologia de uma rede de computadores é caracterizada pela maneira que seus elementos são conectados. A figura 2.4 ilustra algumas das topologias mais comuns: a estrela, anel e barramento [ME84].

Fig.2.4 - Topologias de Redes de Computadores.

Na topologia em estrela, um nó central é ligado ponto-a-ponto a todos os nós da rede. Dessa forma para se estabelecer comunicação entre dois nós é necessário que o nó central faça a intermediação entre eles, criando um circuito ou armazenando e repassando dados ao nó destino. O nó central apresenta dois aspectos negativos para a rede em estrela. Primeiro, representa um gargalo para o sistema, comprometendo o seu desempenho; em segundo lugar, torna a rede de baixa confiabilidade, pois qualquer falha no nó central causa a perda total de comunicação entre os elementos da rede.

A rede de computadores com topologia em anel é constituída por vários nós interligados em série, ponto-a-ponto, formando uma malha fechada. Cada nó é ligado à rede por meio de uma interface especial, repetidor. O repetidor tem a função de ler os dados destinados ao nó, inserir no anel os dados enviados pelo nó e retransmitir os dados que não são destinados ao seu nó. As redes em anel são geralmente unidirecionais, o que simplifica as interfaces. A transmissão de uma mensagem tem início no nó de origem que a insere no anel e transmite adiante até o nó destino. O nó destino lê a mensagem e a retransmite com a informação de que foi recebida para que o nó origem possa remover a mensagem do anel. A rede em anel pode apresentar uma série de problemas: falha de um repetidor, falha ou erro no processamento de mensagem. A falha de um repetidor causa o seccionamento da rede e impede o seu funcionamento. A falha ou erro no processamento de mensagem pode fazer que mensagens circulem indefinidamente pelo anel ou impossibilite que um nó possa transmitir.

Na rede de topologia de barra tem-se um cabo simples ao qual estão ligados os computadores da rede. A transmissão se dá por difusão de um sinal enviado ao cabo como uma sequência simples de pulsos. O uso do meio de transmissão se dá por um método distribuído de controle envolvendo cooperação entre todos os nós da rede. Nesta topologia quando uma estação está transmitindo todas as demais permanecem monitorando o meio de transmissão para identificarem as mensagens destinadas a elas. Dentre as topologias apresentadas é a de melhor confiabilidade.

2.3.4 Protocolos de Controle de Acesso ao Meio de Transmissão

O termo protocolo refere-se ao conjunto de regras e convenções definidas para permitir a comunicação entre computadores de uma rede.

Uma vez que os elementos integrantes de um sistema distribuído compartilham um mesmo meio de comunicação, é fundamental algum tipo de protocolo de controle de acesso ao meio de transmissão. O meio de transmissão deve oferecer um atendimento dinâmico às requisições de acesso. O atendimento pode ser aleatório ou controlado por um algoritmo que defina a sequência dos acessos bem como o tempo em que devem ocorrer [KI86].

O controle de acesso ao meio de transmissão está geralmente associado a topologia e ao meio de transmissão. A seguir são brevemente apresentados os mais frequentes:

Para Redes em Anel:

- Passagem de Permissão ("token")
- Preenchimento de Escaninhos ("Slots")
- Inserção de Registros

Para Rede em Barramento:

- Acesso Múltiplo com Detecção de Portadora ("CSMA")
- Acesso Múltiplo com Detecção de Portadora e Detecção de Colisão ("CSMA/CD")
- Passagem de Permissão em Barramento

A passagem de permissão ("Token Passing") consiste de um pacote de controle que circula pelo anel, "token". Para que uma estação possa transmitir é necessário que ela aguarde um "token" indicando permissão livre, altere o pacote para ocupado e em seguida transmita o seu pacote de dados. Terminada a transmissão de dados a estação retorna um "token" indicando permissão livre. É necessário que o anel tenha algum tipo de mecanismo capaz de resolver erros e circulação indefinida de permissão ocupada.

O preenchimento de escaninhos ("Slots") é baseado na circulação pelo anel de um certo número de escaninhos de tamanho fixo capazes de transportar pacotes de informação. Os escaninhos possuem bits de controle para indicação do seu estado: vazio ou cheio, aceito ou recusado pela estação destino e estação destino ocupada. Para que uma estação possa transmitir é necessário que ela espere por um escaninho vazio, altere o bit de controle para ocupado e então preencha o escaninho com o seu pacote de dados. Terminada a transmissão de dados a estação aguarda o retorno do escaninho com o seu pacote de dados para retirá-lo do anel ou reencaminha-lo dependendo do bit de resposta da estação destino.

A inserção de registro consiste de um registrador de deslocamento de tamanho idêntico ao maior pacote que circula pelo anel. Este registrador está associado a cada estação e exige que a estação tenha um "buffer" para armazenar os pacotes de informações produzidos. Para que uma estação possa transmitir é necessário que ela coloque a informação no registrador, abra o anel e insira o registrador na rede. Isto é feito quando a rede ficar desocupada entre duas mensagens. A mensagem que chega na estação entra pelo registro de deslocamento atrás da mensagem sendo transmitida. Uma nova mensagem só pode ser enviada pela estação quando o registro for retirado do anel. E isto só ocorre quando não há nenhuma mensagem chegando ou o pacote transmitido retorna ao próprio registrador.

O Acesso Múltiplo com Detecção de Portadora ("CSMA") é um protocolo de controle de acesso ao meio de transmissão para redes em barramento. Consiste na capacidade de cada estação em perceber se o meio de transmissão está ou não ocioso. Uma estação para transmitir, observa o meio e transmite se estiver livre, senão aguarda um certo tempo e tenta novamente. Esta situação pode levar a colisões, uma vez que outras estações podem estar também tentando acessar o meio no mesmo instante. Para diminuir as colisões foram propostos três algoritmos de persistência, que são os seguintes:

- *não persistência*: a estação fica em espera por um certo tempo aleatório e volta a sentir o meio de transmissão novamente;
- *1-persistência*: a estação fica monitorando o meio de transmissão até que este fique ocioso e então transmite;
- *P-persistência*: neste caso, a estação também fica monitorando o meio de transmissão até que ele fique ocioso, quando poderá transmitir de acordo com uma probabilidade p .

O Acesso Múltiplo com Detecção de Portadora e Detecção de Colisões ("CSMA/CD") é um protocolo de controle de acesso ao meio de transmissão que tenta solucionar os problemas causados pelas colisões no CSMA. Se ocorre uma colisão, no CSMA, o meio fica inacessível durante todo o tempo de transmissão dos pacotes que colidiram, o que reduz a capacidade do sistema. No CSMA/CD, a estação permanece monitorando o meio de transmissão durante todo o tempo necessário para a transmissão da sua mensagem. Assim se houver uma colisão ela será detectada e a estação interromperá imediatamente a sua transmissão, sinalizando, ainda, a rede que houve uma colisão. Novas transmissões só serão tentadas após o decurso de um tempo aleatório.

A passagem de permissão em barramento é baseada na sequencialização dada as estações formando um anel lógico. Cada estação na rede tem uma posição lógica e conhece a identificação da sua antecessora e sucessora. Um pacote de controle, correspondendo a permissão, passa de estação para estação regulando o direito de acesso ao meio de transmissão. Se uma estação tem a permissão, ela tem o controle do meio de transmissão por um tempo predefinido, podendo transmitir um ou mais pacotes. No final da transmissão a estação passa a permissão para a próxima estação na sequência lógica.

2.4 TÓPICOS DE PROJETO DE SISTEMA DISTRIBUÍDO

Os projetistas de sistemas de computadores que pretendem desenvolver sistemas distribuídos devem abordar os seguintes tópicos no projeto [TA85]:

- primitivas de comunicação
- nomeação e proteção
- gerenciamento de recursos
- tolerância a falhas
- serviços oferecidos

2.4.1 Primitivas de Comunicação

Os computadores que constituem um sistema distribuído necessitam de algum mecanismo para que possam estabelecer comunicação entre eles. Uma vez que normalmente estes computadores não compartilham memória primária, a técnica geralmente empregada para a comunicação é a troca de mensagem (tecnicamente adotada pelo modelo de referência ISO OSI "International Standards Organization - Open Systems Interconnect"). Este modelo está estruturado em sete camadas [TA85]:

- física
- enlace de dados
- rede
- transporte
- sessão
- apresentação
- aplicação

Este modelo é apropriado para a ligação de computadores com variados sistemas operacionais, diferentes códigos de caracteres e arquiteturas. Sua principal desvantagem é a sobrecarga de processamento imposta aos computadores em comunicação [TA85] [RO91].

O modelo mais difundido em sistemas distribuídos que empregam a troca de mensagem é o modelo cliente-servidor. Neste modelo um processo cliente que necessita, por exemplo: ler uma parte de um arquivo, envia uma mensagem deste pedido e aguarda uma resposta do servidor [TA85] [RO91],(Figura 2.5).

Fig.2.5 - Modelo de Comunicação Cliente-Servidor

O processo mais elaborado de troca de mensagem é a chamada de procedimento remoto. O propósito é fazer a semântica de comunicação entre computadores a mais próxima possível das chamadas normais de procedimentos. O seu funcionamento pode ser descrito do seguinte modo: um cliente (programa solicitante) faz uma chamada normal de procedimento, $p(x,y)$ para a sua máquina com o propósito de invocar um procedimento remoto p em uma outra máquina. Um procedimento falso ou ponta ("stub") tem que ser incluído no espaço do programa solicitante ou ligado dinamicamente. Este procedimento que pode ser gerado automaticamente pelo compilador, coleta os parâmetros e forma uma mensagem em um formato padrão. Figura 2.6 [TA85].

Fig.2.6 - Chamada de Procedimento Remoto

A chamada de procedimento remoto requer para a sua implementação um mecanismo de transporte entre o cliente e o servidor. Este mecanismo pode ser [TA85] [RO91]:

- **datagrama**, que consiste de um mecanismo de transporte de mensagem que oferece um serviço não confiável e sem conexão. Não há confirmação da entrega da mensagem, pode haver perda. Um serviço análogo é o telegrama que também não fornece ao remetente confirmação de entrega [TA89].
- **circuito virtual**, que consiste de um mecanismo de transporte de mensagem com controle de fluxo e controle de erro, estabelecido por um canal lógico entre o cliente e o servidor. Um serviço análogo é o circuito físico estabelecido pelo sistema telefônico [TA89].

Como as chamadas de procedimentos locais, as chamadas de procedimentos remotos estão sujeitas a perda de mensagem, falha no servidor, falha no cliente e falha no meio de comunicação. Em caso de falha, outros problemas podem ocorrer além da perda de mensagem, tais como: duplicação de mensagem e bloqueio indefinido do cliente ou servidor. Desse modo há necessidade de mecanismos tanto do lado do cliente quanto do servidor capazes de resolver estes impasses. Uma solução para o problema de bloqueio é a utilização de "timeout" [SV84] [TA85].

2.4.2 Nomeação e Proteção

Nomeação é o processo de mapeamento entre dois domínios, o domínio de objetos físicos e o de objetos lógicos [LE90] [TA85]. O domínio de objetos físicos é constituído dos arquivos, diretórios, processos, serviços, servidores e periféricos de E/S. Em algumas instâncias os nomes destes objetos são cadeias ASCII designadas para uso humano, enquanto que em outras instâncias são números binários para uso interno do sistema operacional. Como exemplo, o sistema de diretório do UNIX utiliza um mapeamento entre os nomes de rotas em ASCII com os números dos i-nodes [TA85].

2.4.3 Gerenciamento de Recursos

O gerenciamento de recursos em sistemas distribuídos é muito mais complexo do que em sistemas centralizados. Isto porque o sistema distribuído não dispõe de tabelas de estado dos recursos em uso como tem os sistemas centralizados. Para efeito de comparação suponha a inicialização de um processo em ambos os sistemas. No sistema centralizado, é necessário simplesmente que o gerente de

processos procure na tabela de processos uma posição vaga para inicializar o novo processo. Já no sistema distribuído torna-se muito difícil localizar um processador livre, principalmente se o projetista do sistema distribuído rejeitou a idéia de ter tabelas de controle centralizadas por razões de confiabilidade.

2.4.4 Tolerância a Falhas

Um sistema tolerante a falhas é aquele que continua funcionando em presença de defeito, mesmo que haja certa degradação [TA85].

Quando se trata de tolerância a falhas dos sistemas distribuídos, deve-se considerar o envolvimento de dois aspectos: a confiabilidade e a disponibilidade. Um sistema é confiável quando não permite corromper ou perder os dados e é disponível quando está livre no instante que for solicitado. O sistema pode tolerar falhas de hardware ou de software, mas em qualquer caso é o software o responsável por contornar o erro. Duas abordagens foram propostas nos últimos anos a fim de tornar os sistemas distribuídos tolerantes a falhas; técnicas de redundância e transações atômicas.

A técnica de redundância é uma técnica simples e eficaz que provê a cada processo um processo "back up" em outro processador. Todos os processos comunicam-se por troca de mensagem. Quando uma mensagem é enviada ao processo é também enviada ao processo "back up". O sistema assegura que ambos receberam corretamente a mensagem. Se um processo falha por defeito de hardware o outro pode continuar. O processo sobrevivente cria um clone seu para resguardar a tolerância a falhas.

Uma transação atômica é uma coleção de mensagens e operações que realizam-se ambas completamente com sucesso ou podem ser retornadas ao estado original como se nenhuma ação tivesse ocorrido [TA85]. Esta técnica como abordagem de tolerância a falhas tem sido conveniente para bases de dados em discos que permite que se faça modificações seguras no próprio disco e de maneira distribuída.

2.4.5 Serviços

A organização do sistema operacional dos sistemas distribuídos em serviços torna o seu núcleo pequeno, confiável e fácil de adicionar, modificar e testar

novos serviços. Um serviço é definido como sendo uma entidade de software em execução em uma ou mais máquinas. Define-se servidor como o software de serviço sendo executado em uma única máquina [SV84]. Os sistemas distribuídos geralmente dispõem de serviços oferecidos pelos seguintes servidores:

- **Servidor de Arquivo.** O servidor de arquivos se constitui no mais importante servidor de um sistema distribuído. Além de permitir o compartilhamento transparente de informações entre muitas máquinas que compõem um sistema distribuído, apresenta características valiosas tais como: mobilidade do usuário, servidor dedicado, implementação econômica do sistema, etc [SV84] [SA90].
- **Servidor de Impressora.** Possibilita aos usuários de sistemas distribuído a utilização de impressora compartilhada. Dispõe de mecanismos que cuidam da armazenagem, sequencialização e impressão dos arquivos solicitados para impressão [TA85] [FU92].
- **Servidor de relógio.** Tem como função a sincronização de cada elemento do sistema [SA90] [RO91]
- **Servidor de Nomes.** Fornece ao sistema um esquema de nomes e endereços, mapeando identificadores em endereços e códigos [SA90] [RO91]

2.5 COMENTÁRIOS

A revisão da literatura permitiu apresentar neste capítulo aspectos relevantes de sistemas distribuídos, tais como:

- características;
- modelos de sistemas;
- subsistema de comunicação;
- topologias de redes;
- protocolos;
- gerenciamento de recursos;
- tolerância a falhas e
- serviços.

Os sistemas distribuídos de interesse abordados nesta revisão, constituem-se de sistemas fracamente acoplados baseados em redes locais. Estes sistemas

oferecem ao usuário um grande potencial colocando a sua disposição recursos transparentes e compartilhados por vários usuários. Estes recursos podem ser diferentes tipos de servidores (impressoras, arquivos, discos óticos, etc) que são acessados através de uma rede de computadores. Esta multiplicidade de recursos torna o sistema distribuído mais confiável, econômico e disponível ao usuário, quando comparado com sistemas centralizados. Assim, um aspecto importante dos sistemas distribuídos em relação aos sistemas centralizados é a tolerância a falhas. No sistema distribuído quando uma parte simples falha o sistema não para e uma outra parte do sistema pode assumir as funções da parte com defeito.

A rede de comunicação, a topologia da rede e o protocolo de acesso ao meio constituem fatores importantes para o bom desempenho do sistema distribuído. O meio de comunicação pode ser dos mais simples como o par trançado ao mais sofisticado como a fibra ótica. A escolha mais adequada do meio de comunicação, da topologia e do protocolo de acesso ao meio, depende dos objetivos propostos para a rede e de fatores tais como: custo, desempenho e alcance.

SERVIDORES DE ARQUIVOS

3.1 INTRODUÇÃO

Os servidores de arquivos merecem, abordagem especial por constituírem a área de atuação do trabalho proposto.

Qualquer que seja a constituição de um sistema de computação, centralizado ou distribuído, o sistema de arquivos desempenha uma importante função do sistema. O sistema de arquivos é um subsistema do sistema operacional que tem o propósito de armazenar informações por longo período. Isto é feito pela implementação de arquivos, que são objetos nomeados que existem a partir da sua criação explícita até a sua remoção e são imunes a falhas temporárias do sistema [LE90].

Um sistema centralizado não oferece dificuldades para um usuário utilizar os serviços do sistema de arquivo. Já o mesmo não acontece quando vários computadores são interligados. Neste caso deve existir algum mecanismo para permitir a um usuário acessar remotamente um arquivo. Para atender a este propósito, três técnicas podem ser consideradas: manter os arquivos separados em cada computador; propor um sistema de arquivo contíguo e a terceira proposta que atende aos sistemas distribuídos, propor um sistema de arquivos global, simples e visível a todos os computadores [TA85].

Pela primeira proposta, um usuário não tem acesso aos arquivos de uma máquina remota via chamadas do sistema. Para acessar um arquivo remoto é necessário que ele utilize um programa de transferência de arquivos, copiando o arquivo desejado da máquina remota para a sua máquina. Um dos exemplos mais

conhecidos que utiliza programa de transferência de arquivos é o programa "uucp" do UNIX.

Pela segunda abordagem um programa pode abrir arquivos em máquinas remotas indicando a máquina e o nome do caminho onde o arquivo está localizado. Como exemplo, suponha um usuário de uma máquina A que queira utilizar um arquivo f em uma máquina remota B. Ele poderia então acessar f por meio de:

```
open("/máquina_B/caminho_de_f",READ);  
open("máquina_B!caminho_de_f",READ);  
ou  
open("../máquina_B/caminho_de_f",READ)
```

A terceira técnica refere-se à abordagem adotada em sistemas distribuídos. Neste caso um sistema global simples é visível por todas as máquinas. Por exemplo, existe um diretório "bin" para os programas binários, um arquivo "password", etc. Quando um programa deseja ler o arquivo "password" ele deve usar uma chamada semelhante a seguinte:

```
open("/etc/passwd",READ)
```

Neste caso não há necessidade de indicar a localização do arquivo. É responsabilidade do sistema operacional localizar e transportar os dados a medida que forem necessários.

O servidor de arquivos se constitui no mais importante servidor de um sistema distribuído. Além de permitir o compartilhamento transparente de informações entre muitas máquinas que compõe um sistema distribuído, proporciona outras características valiosas: [SV84] [SA90]

- **mobilidade do usuário** - permite ao usuário a utilização de seus arquivos em diferentes estações de trabalho. A mobilidade do usuário pode acontecer por vários motivos: falha da sua estação de trabalho, necessidade de trabalhar em outros locais físicos, etc.
- **servidor dedicado** - no servidor de arquivos, um processador dedicado executa todas as funções necessárias ao gerenciamento do serviço de armazenagem e assegura a confiabilidade dos dados. O software das máquinas dos usuários torna-se mais simples, uma vez que não precisa cuidar destes serviços.
- **economia** - é possível utilizar na rede de computadores do sistema distribuído estação de trabalho sem unidade de disco, reduzindo seu custo consideravelmente.

Os sistemas distribuídos que utilizam servidores de arquivos remotos expõe seus usuários a problemas com níveis variados de gravidade de acordo com as aplicações e informações que trafegam pela rede e que são armazenadas nos servidores. Estes problemas são decorrentes de [TA85] [RO91]:

- **segurança.** Como garantir aos usuários privacidade de acesso as informações em redes de computadores? Este é um problema de difícil solução pelas características do próprio meio de transmissão dos sistemas distribuídos. Ele também ocorre com os sistemas centralizados e neste caso é resolvido por meio de uma das duas abordagens: lista de controle de acesso ou nível de capacidade "capability level" dos usuários. Além dessas abordagens os sistemas distribuídos utilizam criptografar o "capability level", porém nada impede que as mensagens possam ser interceptadas por "hackers" e violadas.
- **desempenho.** A questão desempenho está relacionada com a sobrecarga imposta pela rede. Esta sobrecarga tem duas componentes básicas: a primeira diz respeito ao tempo necessário para a rede levar os dados do servidor de arquivos até a estação solicitante; a segunda corresponde ao tempo gasto pela CPU do servidor de arquivos para processar o software de protocolo da rede.
- **confiabilidade.** Esta questão relaciona-se com a qualidade dos meios de armazenagem de informações e da estrutura dos serviços de arquivos. Um procedimento simples para garantir a confiabilidade e disponibilidade das informações, empregado também em todos os sistemas centralizados, baseia-se na cópia periódica dos arquivos ("backup") em fitas magnéticas. A garantia que se tem em caso de falha do hardware é a restauração das informações no instante da última cópia. Consequentemente perdendo-se toda informação armazenada após a última cópia. Este problema constitui o propósito principal do presente trabalho.

Considerando-se o aspecto de tolerância a falhas os arquivos podem ser classificados em robusto, recuperável e normal [SV84]. Um arquivo é dito robusto se resiste a falhas e a deterioração do dispositivo de armazenagem. É recuperável se é possível convertê-lo para um estado anterior consistente, caso as operações de arquivo falhem ou sejam interrompidas pelo usuário. É normal se não dispõe de nenhuma técnica para garantir a integridade.

3.2 CONCEITOS

A maioria dos sistemas distribuídos seguem o modelo cliente-servidor, onde um *servidor* é um subsistema que fornece um tipo particular de *serviço* em atendimento a um pedido de um *cliente* qualquer. *Clientes* são programas que implementam interfaces acessíveis às aplicações de mais alto nível ou acesso direto por pessoa [SV84].

O termo *servidor de arquivos* não se aplica somente a um sistema completo de arquivos, uma vez que muitos servidores de arquivos armazenam simplesmente dados como objetos não estruturados, oferecendo somente algumas primitivas para armazenar e recuperar estes dados [SV84] [SA90]. Existem também servidores de arquivos que implementam sistemas inteiros de arquivos gerenciando arquivos e diretórios [SA90].

Os clientes de um servidor de arquivos pedem e controlam o serviço de arquivo usando um conjunto de operações que formam a *interface de cliente*. Parte do sistema de arquivos pode ser gerenciada pelo servidor de arquivos e parte pela máquina do cliente. A divisão de tarefas entre clientes e servidores é uma decisão importante e geralmente difícil em projetos de sistemas distribuídos. Afim de ilustrar a característica cliente-servidor de sistema distribuído, a figura 3.1 mostra como um subsistema de comunicação permite que um cliente comunique-se com um servidor de arquivos.

Fig.3.1 - Comunicação entre Cliente e Servidor de Arquivos.

3.3 ORGANIZAÇÃO

Em sistemas distribuídos o serviço de arquivos pode ser oferecido por um único ou vários servidores. Santana [SA90a] sugere três abordagens básicas para organização dos serviços de arquivos:

- *centralizado*. Todas as requisições de clientes são atendidas por um único servidor.
- *independente*. Quando o sistema dispõe de um conjunto de servidores de arquivos que trabalham isoladamente.
- *distribuído*. Neste caso existe também um conjunto de servidores de arquivos porém trabalhando em cooperação.

O serviço de arquivos oferecido de maneira centralizada torna o sistema simples e transparente. Este fato porém, torna o sistema frágil sob dois aspectos: desempenho e confiabilidade. O desempenho depende da sobrecarga imposta ao servidor de arquivos, podendo haver degradação se for muito intensa. A confiabilidade é reduzida uma vez que o sistema dispõe de apenas um servidor de arquivos centralizado.

Na organização independente é possível uma distribuição de cargas, porém não é a solução mais adequada. Nesta organização, os servidores de arquivos trabalham isolados, podendo ter alguns sobrecarregados e outros praticamente livres. Além disto, nesta abordagem não é possível obter-se a transparência do sistema.

A organização distribuída dos serviços de arquivos torna o sistema mais confiável e disponível. O aumento da confiabilidade e da disponibilidade do sistema são conseguidos por que o conjunto de servidores de arquivos trabalham em cooperação. Neste caso se um servidor de arquivos falhar um outro poderá assumir o seu serviço. A desvantagem deste método é a exigência de maior complexidade do sistema para manter a consistência entre as várias cópias dos arquivos nos diferentes servidores, arquivos replicados, e, garantir a transparência do sistema.

3.4 ESTRUTURA LÓGICA E FÍSICA

3.4.1 Estrutura Física

Rossato [RO91] apresenta uma estrutura física para servidor de arquivos como sendo uma estação de trabalho que executa o software de serviço de arquivos e dispõe de unidade de disco rígido rápido e de grande capacidade de armazenagem (fig. 3.2).

Fig.3.2 - Estrutura Física do Servidor de Arquivos

3.4.2 Estrutura Lógica

Santana [SA90a], define uma estrutura geral para os servidores de arquivos que é composta de um conjunto de camadas hierárquicas entre o disco físico e o cliente, como é mostrado na figura 3.3.

Fig.3.3 - Camadas Hierárquicas entre o Disco e o Cliente.

Cada camada é responsável por uma série de decisões que dependem da aplicação do servidor de arquivos, do custo, da complexidade, da segurança e da confiabilidade necessárias, etc. A seguir são detalhadas as funções de cada camada:

- A. **Disco físico.** Responsável pela armazenagem dos dados. Pode ser de diferente tecnologia: discos magnéticos (são os mais rápidos e comuns), discos flexíveis (lentos, baratos, pequena capacidade) e discos óticos (baratos, de grande capacidade e alta confiabilidade)
- B. **Controlador de disco.** Negocia com o meio físico de armazenagem. As camadas seguintes vêem o disco como uma lista de blocos numerados de 0 a (nº de blocos - 1). O tamanho do bloco de armazenagem deve ser decidido nesta camada. Blocos grandes podem dar melhor desempenho ao sistema, mas podem ser sub-utilizados. Blocos pequenos podem ter melhor aproveitamento de armazenagem mas podem diminuir o desempenho.
- C. **Cache.** Camada opcional que armazena temporariamente porções de dados que estão correntemente em uso. Tem por função melhorar o desempenho do sistema, reduzindo o número de acessos ao disco físico, o que reduz o tempo de resposta. Os caches podem ser centralizados no servidor de arquivos ou distribuídos nas máquinas de clientes e ou outros servidores.

D. Gerenciamento de memória. É responsável pela alocação de espaço em disco. Seu objetivo principal é a organização eficiente do disco. O gerenciamento de cache, se existir, é realizado também nesta camada.

E. Sistema de arquivos. Duas atividades podem ser identificadas nesta camada. Um sistema de arquivos de cobertura ("flat") responsável pelo reconhecimento dos nomes textuais e gerenciamento do diretório de arquivos para o cliente (realizada pela maioria dos servidores de arquivos), e algumas implementações responsáveis por fornecer ao sistema desempenho, segurança e confiabilidade. Esta camada é responsável também pelas seguintes tarefas:

- *estrutura e gerenciamento de arquivos.* É responsabilidade do sistema de arquivos gerenciar arquivos e manter tabelas para determinação dos endereços dos arquivos em disco.
- *estrutura de diretório.* Responde pela tradução do nome textual do arquivo para um identificador. Geralmente o diretório mantém outras informações a respeito do arquivo, como tamanho e direitos de acesso.
- *melhoria do desempenho, segurança e confiabilidade do sistema.* Já foram abordados anteriormente alguns mecanismos com o propósito de melhorar o desempenho do sistema, dentre eles, a implementação de caches e replicação de arquivos. A segurança pode ser tratada com o controle de acesso e a questão de confiabilidade por meio de transações atômicas e armazenagem estável.
- *implementações para aplicações específicas.* Os servidores de arquivos são utilizados como suporte para aplicação específica, tal como:
 - *sistema de arquivos.* Aplicação típica do servidor de arquivos.
 - *base de dados.* O cliente atualiza e recupera campos de dados de tamanho variável.
 - *memória virtual.* O cliente acessa páginas de memória virtual correspondentes a sua estação de trabalho.

Considerando-se a variedade de serviços do sistema de arquivos, surge uma dificuldade de projeto de servidor de arquivos que consiste em determinar qual parte será de responsabilidade do servidor e qual parte do cliente. É um *problema de*

decomposição. Este problema apresenta dois extremos para a interface de cliente [SA90a] [RO91]:

- *interface de baixo nível*. O servidor de arquivos controla somente o meio físico de armazenagem, enquanto que o sistema de arquivos é implementado no cliente. Neste caso, somente as camadas A, B, C e D são implementadas no servidor.
- *interface de alto nível*. O sistema de arquivos é instalado completamente no servidor com a presença de todas as camadas.

Considerando-se a grande variedade de opções entre estes dois extremos Birrell and Needham [BI80] definiram o *servidor de arquivos universal*. O servidor de arquivos universal pode oferecer flexibilidade suficiente para alguns clientes especiais e ao mesmo tempo facilidades especializadas para outros clientes por meio de uma interface de alto nível [SA90a].

3.5 EXEMPLOS

- *Alpine File System* [SV84], [SA90a] foi desenvolvido no Xerox Palo Alto Research Center e está em operação desde 1983. Projeto para dar suporte a pesquisas em organizações de bases de dados. Alpine é um sistema de arquivos distribuído, que enfatiza a confiabilidade.
- *Amoeba File Server* [TA85], [SA90a] foi desenvolvido na Universidade de Vrije, em Amsterdã, e é um servidor de arquivos rápido, limitado somente pelo meio de comunicação. O Amoeba utiliza um mecanismo para controle de concorrência otimista.
- *Apollo Domain* [NE84], [SA90a] foi desenvolvido pela Companhia de Computadores Apollo (USA) e executa em estações de trabalho Apollo. Um servidor de nomes gerencia o sistema de diretório hierarquico que garante a unicidade dos nomes textuais.
- *CFS (Cambridge File Server)* [SV84], [SA90a] foi desenvolvido na Universidade de Cambridge com o propósito de implementar um servidor de arquivos universal. O servidor de arquivos suporta aplicações de memória virtual e é usado por diversos computadores com sistemas operacionais diferentes.
- *CMCFS (Carnegie Mellon Central File System)* [SV84], [SA90a] foi desenvolvido na Universidade de Carnegie Mellon no início de 1980

e não foi totalmente implementado. O propósito deste sistema é dar suporte a aplicações de base de dados.

- *DCFS (Distributed Caching File Systems)* [SA90a] foi desenvolvido na Xerox Palo Alto Research Center (USA) e auxilia programadores a gerenciar seus ambientes de nomes locais e compartilhar versões consistentes de arquivos. Enfatiza o uso de cache.
- *Felix File Server* [SV84], [SA90a] foi desenvolvido na Bell-Northern Research (USA) com o propósito de suportar aplicações de memória virtual e bases de dados.
- *Keele File Server* [SA90a] foi desenvolvido na Universidade de Keele (UK) e implementado em computadores LSI-115 e PDP-11/34, cada um com sua própria unidade de armazenagem. O desempenho do sistema é melhorado pelo uso de caches e replicação de arquivos.
- *Pulse Distributed File System* [SA90a] foi desenvolvido na Universidade de York (UK) fornece um sistema de arquivos hierarquico do tipo UNIX. O servidor de arquivo mantém várias cópias dos arquivos para melhorar a confiabilidade e desempenho.
- *Sprite File System* [LE90], [SA90a] foi desenvolvido na Universidade da California em Bekerley. Constitue-se de uma coleção de domínios cada qual com uma estrutura de árvore. Os domínios são ligados em uma única hierarquia, ligando as folhas de um domínio com a raiz de outros.
- *Sun Network File System (NFS)* [LE90], [SA90a] fornece acesso remoto sob o ambiente UNIX. O servidor de arquivos é implementado em estações de trabalho SUN.
- *Swallow* [SV84], [SA90a] foi desenvolvido no Laboratório de Ciência da Computação do Instituto de Tecnologia de Massachusetts, e é baseado em um novo modelo de armazenagem de objetos.
- *TRICE File Server* [SA90], [SA90a] foi desenvolvido no Departamento de Eletrônica e Ciência da Computação, Universidade de Southampton (UK) e executa em microcomputadores baseados em M68020. Foi implementado em BCPL e apresenta uma interface de cliente de alto nível.
- *WFS (Woodstock File Server)* [SV84], [SA90a] foi desenvolvido na Xerox PARC para dar suporte ao sistema de escritório Woodstock. Em operação desde 1975, em computadores Alto, recebeu mais tarde uma extensão que passou a fornecer mecanismos para melhorar a confiabilidade do sistema (EWFS).

- *XDFS (Xerox Distributed File System)* [SV84], [SA90a] foi desenvolvido pela Xerox PARC e implementado em linguagem Mesa em computadores Alto. É utilizado para aplicações de bases de dados.

3.6 TÓPICOS DE PROJETO DE SERVIDOR DE ARQUIVOS

Serão apresentados a seguir alguns aspectos relevantes de projeto e organização interna de servidores de arquivos. Inicialmente serão tratados os aspectos de organização, tipos, identificação e operações em arquivos.

3.6.1 Estrutura de Arquivos

Arquivo é uma sequência ordenada de "bytes" que podem ser lidos e ou escritos. Um arquivo tem um nome textual associado e é formado por duas partes: dados e cabeçalho [SV84], [SA90a]. O cabeçalho tem informações sobre o arquivo tais como: data da criação, tamanho do arquivo, data da última modificação, etc. Os dados são divididos e mapeados em blocos de armazenagem. Os blocos podem ser organizados usando diferentes tipos de estruturas de dados [SV84]:

- *árvores de vários níveis*. Os blocos são organizados em uma árvore com um ou vários níveis, dependendo do tamanho do arquivo. Exemplo de sistemas que utilizam esta organização: Amoeba, CFS, CMCFS, EWFS, Felix, e WFS.
- *conjunto de blocos de tamanho fixo*. Páginas de diferentes arquivos são localizadas através de um mapa comum organizado como uma árvore-B que mapeia o par (FID,pag#) diretamente no endereço do bloco particular de armazenagem. Exemplos: XDFS, Alpine e TRICE.

O cabeçalho do arquivo é armazenado normalmente em uma estrutura de dados diferente, podendo ser inserido na própria estrutura do arquivo. Como exemplo, a estrutura usada pelo CFS é uma árvore e o cabeçalho é armazenado na página da raiz [SA90a].

Todo arquivo possui um identificador único (FID) que é usado quando o arquivo é acessado. O FID é gerado pelo sistema no instante da criação do arquivo e só desaparece com a eliminação do arquivo. O FID pode ser criado de várias maneiras:

- *FID dependente do endereço.* não é necessário tabela para determinar o endereço do arquivo. Oferece bom desempenho mas é pouco flexível. Exemplos: CFS e TRICE.
- *FID parcialmente dependente do endereço.* O FID contém parte do endereço de armazenagem. Também apresenta pouca flexibilidade. Exemplos: Alpine, Pulse e XDFS.
- *FID independente do endereço.* necessita de uma tabela para determinar o endereço do arquivo. Apesar de mais complexo é flexível, permitindo mover todo o arquivo com facilidade. Exemplos: Felix e Swallow.

3.6.2 Operações em Arquivos

Para que um cliente solicite um serviço ao servidor de arquivos é necessário que existam algumas primitivas que realizem operações em arquivos. As primitivas essenciais são [SV84]:

- create-file (<pars>)	returns (FID)
- read-data (FID,<selector>)	returns (data)
- write-data (FID,<selector>)	returns (ack)
- delete-file (FID)	returns (ack)

Sendo que:

<> - indica que o parâmetro é opcional;

pars - pode designar o tipo do arquivo quanto a classificação: normal; recuperável ou robusto.

selector - indica a posição de uma parte do arquivo a ser lida ou gravada;

data - unidade de dados lidos ou gravados. Depende da implementação. Pode ser: o arquivo todo (p.ex.: Swallow); uma página (p.ex.: WFS, XDFS, Amoeba e Felix); e uma parte do arquivo (p.ex.: CFS e CMCFS).

ack - retorna a indicação do sucesso ou fracasso da operação.

Um servidor de arquivos deve suportar execução atômica de operações compostas que permitem múltiplos acessos a um mesmo arquivo ou múltiplos acessos a múltiplos arquivos. Esta composição de operações é chamada transação. Uma transação é uma coleção de mensagens e operações que realizam-se completamente com sucesso ou podem ser retornadas ao estado original como se nenhuma ação tivesse ocorrido [TA85] [SV84]. As primitivas de uma transação podem ser:

begin-trans(<pars>)	return(TID)
end-trans(TID)	return(ack)
abort-trans(TID)	return(ack)

Sendo que:

TID - identificador único da transação.

Uma transação pode ser abortada por uma falha no servidor de arquivos ou pelo próprio servidor. Neste caso a primitiva end-trans do cliente falha.

As operações elementares em arquivos devem ser estendidas e incluir a TID:

- create-file(TID,<pars>)	returns(FID)
- read-data(TID,FID,<selector>)	returns(data)
- write-data(TID,FID,<selector>,data)	returns(ack)
- delete-file(TID,FID)	returns(ack)

3.7 CONTROLE DE ACESSO

É necessário que os sistemas distribuídos forneçam mecanismos capazes de garantir o controle de acesso aos arquivos [SV84], [TA85]. O controle de acesso pode ser implementado por meio de "capability", identidade e criptografia.

- *"Capability"* [SA90a], [RO91]. O cliente deve possuir uma "capability" válida para ter acesso ao arquivo. A maioria dos sistemas utilizam o FID como "capability", uma vez que é único no sistema. Exemplos de sistemas que utilizam esta técnica: Amoeba, CFS, e Felix.
- *Identidade* [SA90a], [RO91]. Neste caso, o cliente deve apresentar além do FID, uma prova de identidade. O sistema dispõe de uma lista de controle de acesso que fornece os direitos de cada cliente para cada arquivo. Exemplos de sistemas que utilizam esta técnica: Alpine, CMCFS, Felix, TRICE e XDFS.
- *Criptografia* [SA90a], [RO91]. Cada arquivo tem uma chave única e é armazenado e transferido de forma criptografada usando esta chave. Exemplo de sistema que utiliza esta técnica: Swallow.

3.8 CONFIABILIDADE

A confiabilidade é um aspecto dos sistemas distribuídos tolerante a falhas. Um sistema é confiável quando não permite corromper ou perder os dados em presença de falhas dos servidores de arquivos [TA85]. A abordagem mais simples sugerida por Tanenbaum [TA85] é: projetar o sistema cuidadosamente, usar discos de melhor qualidade e fazer cópias ("back up") periodicamente. Se um disco apresentar defeito de hardware, todo trabalho de armazenagem de arquivos realizado desde a última cópia será perdido.

Este método pode parecer assustador à primeira vista, mas quase todos os sistemas de computadores centralizados trabalham desse modo e como o MTBF ("Mean Time Between failure" - tempo médio entre falhas) para os discos atuais é da ordem de 50.000 horas, isto funciona muito bem [TA85].

Alguns sistemas distribuídos oferecem às aplicações que exigem maior grau de confiabilidade, serviço de arquivo robusto. Uma abordagem simples é dada pelo uso de disco imagem: Cada solicitação de **write-file** é realizada em paralelo em duas unidades de disco. A todo instante os dois discos são idênticos e qualquer um pode substituir o outro em caso de falha [TA85]. A replicação de arquivos também é uma boa técnica para melhorar a confiabilidade. Levy [LE90] ressalta que a replicação de arquivos deve ser feita em máquinas diferentes e não em diferentes meios de armazenagem. Justifica que a replicação em várias máquinas beneficia o desempenho do sistema.

O refinamento desta técnica permite que o servidor de arquivos ofereça armazenagem estável (qualquer mídia, como discos replicados, que resistem a falhas) e transações atômicas. As transações atômicas são apropriadas para [RO91]:

- *falhas*. As operações que compõe a transação são todas executadas completamente, ou nenhuma é executada;
- *acesso concorrente*. As transações concorrentes devem produzir o mesmo resultado caso fossem executadas em série.

A execução de uma transação atômica pode ser dividida em três partes [SA90a]:

- *escalonamento*. Define a intercalação das requisições segundo mecanismos de controle de concorrência;
- *operações*. As requisições são executadas. As operações de atualização introduzem dificuldades caso ocorra alguma falha. Neste caso o

arquivo deve ser restaurado ao seu estado anterior. O arquivo deve ser recuperável.

- *encerramento*. Este é o problema mais difícil da implementação de transações atômicas, pois é necessário garantir que todas as modificações sejam executadas corretamente e de forma permanente antes de encerrar a transação.

[RO91]: Os mecanismos utilizados no escalonamento são os seguintes [SA90a],

- *bloqueio*. É usado para garantir o acesso exclusivo a uma unidade de dados. Exemplos de sistemas que utilizam este mecanismo: Alpine, CMCFS, EWFS, Felix e XDFS.
- *"timestamp"*. Um número único relacionado com o tempo com o propósito de ordenar as transações. Exemplo de sistema: Swallow.
- *otimista*. Considera que não acontecerá nenhum conflito de acesso. Exemplo de sistema: Amoeba.

O mecanismo de controle de concorrência deve ser capaz também de resolver problemas de "deadlock", evitando que uma transação fique aguardando serviço por tempo indeterminado. Alguns sistemas como Alpine e Felix implementam mecanismos de detecção de "deadlock" e o sistema Felix implementa mecanismo de prevenção. Outros sistemas como XDFS, EWFS utilizam tempo limite para a realização da transação.

A técnica básica para assegurar a recuperabilidade de um arquivo é evitar a sobre-escrita dos dados atuais na armazenagem física [SV84]. Essas técnicas podem ser [RO91]:

- *"shadow-page"*. As páginas alteradas são escritas nos blocos livres. Quando todas as páginas foram gravadas corretamente, a raiz do arquivo é atualizada. Exemplos de sistemas que utilizam esta técnica: EWFS e Felix.
- *"undo/redo log"* É mantido pelo servidor de arquivos um registro ("log") das alterações do arquivo. Em caso de falhas, é possível reverter as alterações. Um exemplo de sistema que utiliza este método é o Alpine.
- *Registro ("log") de intenções*. Neste caso é mantido pelo servidor de arquivos um registro de intenções de mudanças, que serão executadas no encerramento da transação. Exemplo de sistema que utiliza esta técnica: XDFS.

- *versões tentativas*. Um **write-file** cria uma versão tentativa do arquivo todo. Se a transação falhar, a versão tentativa é ignorada. Exemplo de sistema: Swallow.

3.9 COMENTÁRIOS

Este capítulo ressalta a importância dos servidores de arquivos para os sistemas distribuídos, abordando aspectos referentes a: características, estruturas, organização, serviços, aspectos de segurança e confiabilidade. E para efeito de ilustração da aplicação destes aspectos, foram apresentados vários exemplos de implementações de servidores de arquivos.

O servidor de arquivos representa para os sistemas distribuídos seu servidor mais importante, pois trata da parte mais valiosa do sistema, as informações. Além de permitir o compartilhamento transparente de informações entre muitas máquinas que compõe um sistema distribuído, proporciona outras características importantes tais como: mobilidade do usuário, servidor dedicado, economia, etc.

A estrutura de um servidor de arquivos é responsável pela qualidade e complexidade da interface de cliente. A estrutura enfatizada nesta revisão bibliográfica foi a sugerida por Santana [SA90a], que utiliza camadas hierárquicas entre o disco e o cliente. Cada camada é responsável pelas decisões necessárias que dependem da utilização, ambiente e complexidade do servidor de arquivos. Um problema decorrente da estrutura do servidor de arquivos é a decomposição, isto é, a divisão de que trata do que é responsabilidade entre cliente e servidor.

Uma vez que o propósito do servidor de arquivos é oferecer um serviço particular em atendimento a um pedido de um cliente, é necessário que se tenha primitivas que realizem operações, tais como: create-file, read-data, write-data e delete-file.

Algumas técnicas para aumentar a segurança e permitir concorrência nos servidores de arquivos foram consideradas neste capítulo, como por exemplo: mecanismos de transações atômicas e utilização de armazenamento estável. E para melhorar a confiabilidade dos servidores de arquivos foram propostos procedimentos simples como fazer cópias periódicas até os mais elaborados que envolvem arquivos robustos, por meio de técnicas de replicação dos arquivos.

SISTEMAS DE ARQUIVOS BASEADOS EM DISCOS ÓTICOS WORM

4.1. INTRODUÇÃO

Os sistemas óticos de armazenagem de informação, apesar de serem resultado de uma tecnologia recente, estão sendo muito utilizados nas grandes organizações com a finalidade de minimizar a problemática dos arquivos de papel, consulta a microfilmes, discos magnéticos e segurança.

Segundo a "Association for Information and Image Management (AIIM)" dos Estados Unidos, 5% de toda a informação disponível no mundo estarão em discos óticos até o final da década de 90. Apesar de ser um índice muito pequeno quando comparado com o índice de informações armazenadas em papel que é de 95%, não deixa de ser significativo quando comparado com outras tecnologias com mais de 20 anos. Segundo a própria AIIM [OL91], apenas 1% das informações são armazenadas em discos magnéticos e 4% em microfilmes.

Apesar do custo de armazenagem de informação variar em função do tipo de aplicação, os sistemas óticos são os mais baratos, o que estimula o seu uso pelas empresas preocupadas com a armazenagem e recuperação de informações. Um estudo realizado pelos Fuzileiros Navais (Marine Corps) para saber qual o custo para armazenagem de 1 Megabyte de informação, apresentou o seguinte quadro publicado no periódico americano "Government Computer News", em 22 de abril de 1991 [OL91]:

Mídia	US\$/mb
Disco Rígido Removível	15.00
Disco Rígido Fixo	10.00
Papel	4.00
Disquete	1.70
Microficha	0.75
Fita Magnética	0.25
WORM	0.13
CD-ROM	0.02

Os discos óticos estão disponíveis em três categorias, apresentadas a seguir:

- **CD-ROM**, Compact Disc - Read Only Memory - utiliza princípio idêntico ao disco compacto musical para armazenar sons, com a vantagem de armazenar imagens e informações. A diferença tecnológica entre eles é que o CD-ROM usado no sistema de computação não precisa transformar o sinal digital para analógico. O processo de gravação é o de masterização, i.e., o processo industrial que utiliza uma matriz para prensagem do disco,
- **WORM**, Write Once Read Many - foi o primeiro disco ótico desenvolvido diferindo do CD-ROM pelo fato de que a gravação é realizada pelo próprio usuário, e é permanente, i.e. uma vez que uma região do disco é alterada por um processo de gravação, ela não pode mais ser alterada. Esta característica ressalta a importância deste dispositivo como medida de segurança de dados em sistemas de redes de computadores,
- **REGRAVÁVEL**, - este tipo é fabricado utilizando-se duas tecnologias para armazenagem de informação: Alteração de fase e a optomagnética. A alteração de fase tem por princípio básico a mudança na estrutura molecular de uma forma amorfa para uma cristalina, ou vice-versa, permitindo que o disco seja gravado, apagado e regravado com novas informações por inúmeras vezes. A gravação utilizando a tecnologia optomagnética é caracterizada pela gravação no sentido vertical de um disco magnético para proporcionar aumento na densidade de gravação. O processo de gravação e leitura tem certa semelhança com o processo de disco magnético. Porém, utiliza feixe de laser para gravar e ler as informações contidas nos discos. Como o processo é caracterizado pelas alterações magnéticas do disco é pos-

sível gravar, ler, apagar e regravar novas informações. A princípio este disco poderia perfeitamente substituir o disco magnético pois tem maior capacidade de armazenagem por unidade de área, menor custo e maior resistência ao meio. Porém, tem uma desvantagem muito séria, principalmente para sistemas distribuídos, onde o desempenho do disco é fundamental. Esta desvantagem diz respeito ao tempo médio de acesso do disco ótico que é muito elevado quando comparado com o disco magnético.

As vantagens do disco ótico em relação a outros dispositivos, principalmente em relação ao disco magnético são [FU84] [CH86]:

- baixo custo por byte armazenado, cerca de 100 vezes menor que o custo do magnético,
- grande capacidade de armazenagem de informação por mesma unidade de área, cerca de 10 vezes maior que um similar magnético,
- um disco ótico pode ser reproduzido, juntamente com as informações armazenadas no mesmo, em série através de processo industrial de baixo custo enquanto que o magnético a duplicata tem que ser feita uma de cada vez,
- o disco ótico pode ser removido da sua unidade permitindo que se seja usado como arquivo de armazenagem. Por outro lado, a maioria dos discos magnéticos de grande capacidade são fixos na unidade, necessitando que sejam copiados para fitas para efeito de arquivo de armazenagem,
- no caso particular do disco ótico WORM, as informações uma vez escritas no disco não podem mais ser apagadas,
- os discos óticos apresentam maior estabilidade e confiabilidade quanto a resistência ao ambiente tais como: temperatura, umidade e principalmente perturbações magnéticas [MI88].

As principais desvantagens do disco ótico em relação ao magnético são:

- tempo médio de acesso que está entre 50 a 180 milissegundos enquanto que o magnético está entre 8 a 20 milissegundos,
- falta de padronização de protocolo para gravação de informações nos discos óticos WORMs e regraváveis.

4.2. DISCOS ÓTICOS: CARACTERÍSTICAS FÍSICAS

A unidade de disco ótico graças ao seu tamanho reduzido e a sua construção possibilita um baixo custo e excelente confiabilidade operacional. A Figura 4.1 mostra aspectos gerais de uma unidade de disco ótico [FU84].

Fig.4.1 - Aspectos Gerais de uma Unidade de Disco Ótico

A Figura 4.2 ilustra por meio de um diagrama de blocos as partes principais de uma unidade de disco ótico e o seu interfaceamento com um computador [MI88]. A interface SCSI "Small Computer System Interface" padronizada pelo comite técnico ANSI "American National Standard Institute" permite que qualquer dispositivo de memória baseado em rotação tais como discos magnéticos, fitas magnéticas e discos óticos sejam endereçados como unidades lógicas genericas, tornando as diferenças tecnologicas entre estes dispositivos invisíveis ao software controlador do sistema operacional [FU84]. Uma única interface SCSI pode controlar até sete unidades lógicas genéricas ligadas a um computador. A utilização de memória "buffer" pelo disco ótico possibilita o processamento de um grande volume de informações, reduzindo dessa forma a carga do computador. As partes restantes são responsáveis por converter os comandos do computador em ações físicas na unidade

e apropriados aos propósitos de arquivos. A informação é lida pela alteração de amplitude no feixe refletido.

- *Materiais de alteração de fase* - O feixe de laser, neste caso, causa alteração física localizada dentro do filme fino, geralmente uma alteração na estrutura molecular de uma forma amorfa para uma cristalina, ou vice-versa. A leitura da informação se dá por meio das alterações de reflexividade do filme.

A Figura 4.3 mostra uma maneira possível de gravar informações em discos óticos. O feixe de laser que sai da cabeça de gravação e leitura é direcionada para o lado inferior do disco e atravessa o substrato - camada transparente de plástico ou vidro, penetra na camada fina de polímero e bate na camada metálica. Lentes focalizam o feixe assim que atingem a camada metálica, levando seu diâmetro de 2 milímetros para 1 micron. O calor do laser vaporiza uma pequena quantidade do polímero, forçando a camada metálica para cima, formando uma bolha [FU84].

Fig.4.3 - Processo de Gravação de um Disco Ótico

A camada de substrato, de aproximadamente 1,2 milímetros de espessura, de material transparente: plástico ou vidro, tem a função de proteger a camada metálica de gravação de danos físicos que possam ser causados pelas cabeças de leitura e gravação, como também da contaminação por sujeiras. Enquanto que nos discos magnéticos a distância das cabeças à superfície do disco é importante para o seu desempenho e é de alguns microns, nos discos óticos essa distância chega a alguns milímetros e não causa perda de desempenho. Esta característica possibilita a remoção do disco da unidade [FU84].

4.3. DISCOS ÓTICOS: CARACTERÍSTICAS LÓGICAS

Os discos óticos são organizados em unidades chamadas blocos, que são unidades indivisíveis e contínuas de armazenagem. Quando um bloco é gravado no disco, o código de correção de erro correspondente ao bloco também é gravado em uma posição convencional. Por esta razão, blocos individuais do disco, uma vez gravados, nunca são regravados, mesmo que ainda tenha espaço no bloco [GA88]

Os controladores de discos óticos tratam de maneira diferente os blocos livres e os blocos já gravados. Uma vez que somente os blocos já gravados possuem código de correção de erro, uma tentativa de leitura de um bloco livre retorna uma indicação de erro. Um benefício importante para aplicações práticas.

O processo de atualização de dados de sistemas de arquivos de discos magnéticos consiste em regravar os blocos que contêm estes dados no disco. Arquivos são facilmente eliminados de diretórios, regravando-se o diretório sem os nomes dos arquivos eliminados. A área de dados ocupada pelo arquivo pode ser recuperada, retornando-se os blocos utilizados pelo arquivo ao conjunto de blocos livres do sistema de arquivos. Já a atualização dos arquivos no sistema baseado em disco ótico de única escrita é mais complicada. O bloco físico não pode ser alterado, dessa forma, arquivos e diretórios que são logicamente alterados devem ser regravados em novos blocos livres do disco. Dessa maneira o sistema de arquivos de disco ótico guarda a versão mais recente de cada diretório e arquivo [GA91].

A armazenagem de arquivos em discos óticos pode ser feita por meio das seguintes abordagens: anexar no final, que consiste em gravar novos dados após os últimos gravados; copiar o arquivo todo para o disco toda vez que o arquivo for alterado; armazenar os dados no disco ótico e informações auxiliares em discos magnéticos e usar árvores B para aplicações de base de dados [GA88].

O método de anexar no final é adequada para os arquivos em que novos dados são acrescentados ao final do arquivo, tal como, arquivos de registro "log files". O método de copiar é adequado aos arquivos que são raramente atualizados. Na abordagem disco ótico com auxílio de disco magnético, dados não organizados são armazenados no disco ótico e a estrutura de dados necessária para tornar o disco ótico compreensível é armazenada em disco magnético. Este método tem um sério inconveniente, uma falha no disco magnético causará a perda dos dados armazenados no disco ótico. O último método usando árvore B para organizar os dados no disco ótico utiliza de maneira inadequada o disco.

4.4 EXEMPLOS DE SISTEMAS DE ARQUIVOS UTILIZANDO DISCOS ÓTICOS

Considerando-se o levantamento bibliográfico realizado, identificou-se duas propostas de sistemas de arquivos utilizando discos óticos que merecem destaques:

- Sistema de arquivo de única escrita ("WOFS") [GA91]
- Sistema de arquivo de acesso aleatório para discos óticos de única escrita ("OFC") [GA88].

A primeira proposta refere-se ao projeto de pesquisa desenvolvido por Garfinkel [GA91], em 1985 no MIT Media Lab. Trata-se de um sistema de arquivos projetado para discos óticos de única escrita. Como já foi abordado anteriormete, este disco tem uma característica singular: uma vez que um bloco é gravado no disco WORM, ele não pode mais ser alterado. Os discos WORMs oferecem armazenagem permanente, contudo os sistemas operacionais como UNIX e MS-DOS necessitam de software especial para poder utilizá-los. Daí o empenho de Garfinkel [GA91] em descrever o projeto e evolução de um componente de software, conhecido como "Sistema de Arquivo de única Escrita". O objetivo deste projeto é composto por duas partes:

- desenvolvimento de um sistema de arquivo padrão que define a maneira de organizar informações dos arquivos no disco ótico;
- implementação do sistema de arquivos através de funções de biblioteca escritas na linguagem C.

Um aspecto relevante deste projeto é que o sistema de arquivo foi projetado para ser independente do sistema operacional. Desse modo arquivos gravados em discos óticos em um sistema operacional podem ser lidos por outro sistema operacional. Nos últimos cinco anos o sistema passou por três modificações de projeto. A última modificação permitiu que o sistema de arquivos fornecesse muitas chamadas de funções, incluindo *open()*, *close()*, *read()*, *write()*. Passou também a permitir que o usuário tivesse acesso a versões dos arquivos.

A estrutura básica para gravação dos dados proposta por Garfinkel [GA91] para o seu sistema de arquivo de disco ótico consiste de:

- gravar os dados no sistema de arquivo de disco ótico através de uma sequência de operações **block-write()**, começando com o primeiro bloco físico do disco e continuando até o último bloco do disco;

- dividir o disco em duas regiões: uma em que os blocos estão sendo gravados e outra em que os blocos estão livres. O ponto de transição entre essas regiões é chamado de "último bloco gravado".

O sistema não dispõe de região específica para diretórios ou apontadores de arquivos (como é o caso dos i-nodes do UNIX). Isto elimina um problema comum que ocorre com outros sistemas: parte do disco enche não permitindo mais o seu uso, mesmo que a outra parte esteja parcialmente cheia.

O sistema WOFS utiliza um processo otimizado baseado em busca binária para localizar o último bloco gravado.

A interface do sistema operacional com o sistema WOFS é feita de maneira transparente tanto para o MS-DOS como para o UNIX. A implementação no MS-DOS utiliza um programa do tipo termina e fica residente (TSR) que intercepta a interrupção dos serviços do DOS. Cada interrupção é analisada para saber quem deve atendê-la, o WOFS ou o MS-DOS. Para o sistema UNIX foi desenvolvido um servidor com o WOFS baseado no NFS ("Network File System") da Sun Microsystems [SU90]. O servidor traduz todos os requisitos da rede para as operações apropriadas do WOFS.

A segunda proposta é a de um gabinete de arquivos ótico, um sistema de arquivos de acesso aleatório para discos óticos de única escrita (OFC), desenvolvido por Gait [GA88]. O propósito deste sistema é o de armazenar todas as versões dos dados, como em um escritório convencional, uma vez que as versões antigas podem ser necessárias por razões históricas ou legais.

O sistema OFC usa uma estrutura de dados que pode ser gravada no disco ótico. Esta estrutura pode ser incrementada e alterada dinamicamente se os arquivos são alterados. Conserva o espaço em disco através de atualizações baseadas em bloco a bloco e usando estratégias de cópia de cache. Embora a mídia de única escrita seja estável, o sistema dispõe de procedimentos de recuperação de dados do disco ótico em caso de falha de CPU e ou controlador.

A interface com o sistema operacional preserva a compatibilidade das estruturas dos sistemas de arquivos de outros sistemas operacionais, aparentando ser um disco magnético convencional.

O método utilizado pelo OFC para armazenagem dos dados no disco ótico é muito próximo do método de somente copiar, visto que os arquivos são armazenados no disco ótico e podem ser arbitrariamente atualizados, juntamente com toda informação da estrutura organizacional. Contudo o sistema OFC é eficiente em termos de utilização do disco e desempenho, porque utiliza um processo de atualização baseado em bloco por bloco.

O sistema operacional trata o disco lógico como uma sequência linear de blocos. O sistema OFC preserva a interface esperada pelo sistema operacional de disco lógico criando uma ordem linear em forma de folhas de uma árvore. O acesso ao disco lógico é feito em forma de uma árvore de arranjo de ponteiros chamada árvore do sistema de arquivos ("FST"), onde as folhas contém os blocos do disco lógico necessárias ao sistema operacional. O bloco raiz da FST é sempre um bloco de ponteiros que apontam para o segundo nível de blocos do disco. O segundo nível de blocos contém ponteiros de níveis adicionais, blocos de disco lógico ou blocos livres. A página de ponteiros de 32 bits pode endereçar até quatro bilhões de blocos de disco lógico ou quatro terabytes de dados armazenados em disco ótico. Esta técnica permite calcular com eficiência a posição física no disco ótico de qualquer bloco do disco lógico. Esta eficiência é conseguida pelo uso de cache para os nós dos ponteiros e de cálculo dos deslocamentos dos ponteiros por operações de mascaramento e deslocamento de bits.

Além dos exemplos apresentados acima, um outro projeto que faz uso dos discos óticos de única escrita é o SWALLOW [SV81]. O sistema SWALLOW utiliza os discos óticos para registro "log" das alterações dos arquivos como parte do sistema de arquivos.

4.5 COMENTÁRIOS

Este capítulo apresentou as principais características de discos óticos, incluindo vantagens e desvantagens de utilização, estrutura física e lógica e exemplos de aplicações de discos óticos de única escrita em sistemas de arquivos.

Inicialmente foram abordados aspectos relevantes da estrutura física dos discos óticos tais como, materiais que os constituem, forma de gravação e leitura de dados e resitência ao meio ambiente. Mostrou-se em seguida que os discos óticos são organizados em unidades chamadas blocos, que são unidades indivisíveis e contínuas

de armazenagem. Quando um bloco é gravado no disco, o código de correção de erro correspondente ao bloco também é gravado em uma posição convencional. Por esta razão, blocos individuais do disco, uma vez gravados, nunca são regravados, mesmo que ainda tenha espaço no bloco

Foram tratadas também algumas técnicas propostas para a armazenagem em discos óticos, que podem ser: somente anexar no final; somente copiar; armazenar os dados no disco ótico e informações auxiliares em discos magnéticos e a última empregar técnicas de árvores B para aplicações de base de dados.

Os exemplos apresentados de sistemas de arquivos utilizando os discos óticos de única escrita demonstram a viabilidade desses recursos e servem de estímulo para o desenvolvimento do trabalho proposto.

PROPOSTA DE UM SISTEMA AUXILIAR DE SEGURANÇA DE ARQUIVOS

5.1 INTRODUÇÃO

Um dos problemas de segurança de informações existente em qualquer sistema computacional é a confiabilidade. Este problema teve algumas soluções propostas para os sistemas distribuídos, dentre elas, uma de baixa complexidade que consiste em fazer cópias periódicas dos sistemas de arquivos ("backup's"); uma outra solução envolve arquivos robustos que utiliza a técnica de replicação criando arquivos imagem ("mirror disk") e uma terceira solução, que é um aperfeiçoamento da segunda, permite que o servidor de arquivos ofereça armazenagem estável e transações atômicas [TA85]. A primeira abordagem deixa totalmente a responsabilidade de copiar e restaurar o sistema de arquivos para o usuário. Se o período entre as cópias for muito longo e se o servidor de arquivos falhar as informações armazenadas após a última cópia serão perdidas. A segunda e terceira já são mais elaboradas, mais caras e tornam a estrutura do servidor de arquivos mais complexa.

Santana, em [SA90] propõe a utilização de um servidor de arquivos com maior capacidade de armazenagem, através da utilização de discos óticos (esta abordagem é discutida na seção 5.4).

A proposta deste trabalho é baseada nesta abordagem e propõe a utilização de disco ótico de única escrita como servidor auxiliar do servidor de arquivos levando em consideração aspectos importantes tais como: a mídia do disco ótico é durável [GA88]; baixo custo por byte armazenado [OL91]; grande capacidade de armazenagem [GA88] e armazenagem de versões de arquivos. Além destes

aspectos levou-se em consideração a lacuna existente entre as soluções propostas para confiabilidade dos servidores de arquivos. Desse modo pretende-se propor mecanismos que possam cuidar da cópia de arquivos para os discos óticos automaticamente. Deverão ser considerados procedimentos que monitorem os arquivos, agindo em função da data de criação, última atualização e longevidade do arquivo [SA87] [SA81] ou então por solicitação privilegiada do cliente através de chamadas de procedimentos. Em caso de falha ou solicitação do cliente, a restauração das informações, será também automática.

A implementação dos procedimentos propostos foi executada na linguagem de programação C e em um ambiente baseado no sistema operacional MINIX [TA85], compatível com o UNIX versão 7. A razão da escolha deste sistema deve-se ao fato de que o MINIX está estruturado de maneira modular, como uma coletânea de processos que se comunicam entre si e com os processos dos usuários através de mensagens. Quando o sistema é compilado, quatro programas binários independentes são criados: o núcleo (incluindo os "drivers"), o gerenciador de memória, o sistema de arquivos e init (que lê /etc/ttys e cria os processos "login"). Desse modo é possível recompilar uma única parte sem alterar as demais, por exemplo: o sistema de arquivos. A seção 5.3 deste capítulo apresenta características gerais do sistema MINIX.

A fim de simplificar a referência ao sistema desenvolvido, será utilizada, quando conveniente, a designação **saa-WORM** em substituição a "sistema auxiliar de arquivos WORM".

5.2 IMPORTÂNCIA DO SISTEMA AUXILIAR DE ARQUIVOS

A importância do sistema auxiliar de arquivos, como já ressaltado por diversas vezes neste trabalho, se deve à preocupação com questões de segurança, disponibilidade e integridade dos dados. Além dessas questões, outra de grande relevância é que trata da disponibilidade de espaço para os sistemas de arquivos nos meios magnéticos. A atenção nesse caso se volta aos arquivos que permanecem nos sistemas sem serem acessados por vários meses, não importando tipo ou proprietário do arquivo, mas sim o comprometimento de espaço.

Análise do uso do armazenamento de apoio no sistema TRICE [SA87] mostrou que os arquivos produzidos em um dia normal de trabalho são parcialmente removidos em um curto espaço de tempo. Após dez dias, cerca de 50 por cento dos

arquivos originais ainda permanecem no sistema. Entre duas a quatro semanas é esperado que apenas 30 por cento continuem no sistema. Isto sugere que arquivos com mais de um mês dificilmente sejam modificados ou removidos.

Com o propósito de ressaltar o comprometimento de espaço por arquivos que não são acessados por vários meses e como se dá a sua distribuição em função do tamanho, realizou-se um levantamento em três sistemas de computação. Neste levantamento, quantificou-se o número total de usuários, área total dos sistemas de arquivos e a área total comprometida com arquivos não acessados a mais de 180 dias. A escolha do fator 180 dias foi arbitrária, julgando-se suficiente para ressaltar o comprometimento de espaço por esses arquivos. Além do que, este fator de 180 dias permite definir um fator menor tal como 90 dias para remoção destes arquivos para um outro meio qualquer, liberando seus espaços nos meios magnéticos. É evidente que todos os arquivos que não são acessados a mais de 180 dias, também não foram acessados quando completaram 90 dias. Assim poderiam ter sido removidos aos 90 dias sem acesso e teriam assim, deixado livre seus espaços por pelo menos mais 90 dias.

Os três sistemas de computação utilizados no levantamento foram os seguintes:

A - Estação de trabalho Intergraph 220 do Centro de Apoio à Informática da Faculdade de Medicina de Ribeirão Preto - USP;

B - Estação de trabalho SUN do Laboratório de Física da Faculdade de Filosofia Ciências e Letras de Ribeirão Preto - USP e

C - Computador CDC 4360 do Centro de Computação Eletrônica - USP.

É importante ressaltar que o levantamento realizado limitou-se ao acesso permitido ao autor como usuário normal destes sistemas, com exceção do sistema A, no qual o autor tem privilégio de super usuário. Portanto os valores aqui apresentados estão limitados. A tabulação dos resultados obtidos do levantamento realizado nos três sistemas é apresentado nas tabelas 1 a 4. As medidas das áreas são dadas em blocos de 512 bytes para os sistemas A e C, e de 1024 bytes para o sistema B. As classes são dadas em número de blocos e a percentagem reflete a área comprometida pelos arquivos não acessados a mais de 180 dias sobre a área em uso.

Tabela 1. Comprometimento do espaço dos sistemas de arquivos com arquivos não acessados a mais de 180 dias.

Sistema	Número de Usuários	Área Total (blocos)	Área em Uso (blocos)	Área Comprometida (blocos)	%
A	53	236.602	204.994	170.244	83,05
B	49	140.638	106.572	84.769	79,54
C	715	558.438	435.727	187.026	42,92

Tabela 2. Comprometimento do espaço do sistema de arquivos (/usr), do sistema A, em função de classes de arquivos não acessados a mais de 180 dias.

	Classe (blocos)	Número de Arquivos	Área Comprometida	% (Total em Uso)
	- 2	1.893	2.847	1,39
	2 - 10	2.385	12.063	5,88
	10 - 50	1.410	34.733	16,94
	50 - 250	418	47.132	22,99
	250 - 500	70	23.572	11,50
	500 - 1000	48	33.204	16,20
	1000	11	16.693	8,14
Total		6.235	170.244	83,05

Tabela 3. Comprometimento do espaço do sistema de arquivos (/usr), do sistema B, em função de classes de arquivos não acessados a mais de 180 dias.

	Classe (blocos)	Número de Arquivos	Área Comprometida	% (Total em Uso)
	- 2	5.149	6.551	6,15
	2 - 10	2.552	13.330	12,51
	10 - 50	1.005	20.381	19,12
	50 - 250	180	18.203	17,08
	250 - 500	23	7.632	7,16
	500 - 1000	13	9.552	8,96
	1000	7	9.120	8,56
Total		8.929	84.769	79,54

Tabela 4. Comprometimento do espaço do sistema de arquivos (/usr), do sistema C, em função de classes de arquivos não acessados a mais de 180 dias.

Classe (blocos)	Número de Arquivos	Área Comprometida	% (Total em Uso)
- 2	1.203	1.515	0,35
2 - 10	1.878	10.588	2,43
10 - 50	1.410	29.964	6,88
50 - 250	362	37.479	8,60
250 - 500	39	12.984	2,98
500 - 1000	74	51.768	11,88
1000	25	42.728	9,81
Total	4.991	187.026	42,92

Nos três sistemas de computação, o comprometimento de espaço dos sistemas de arquivos foi expressivo como pode ser observado pelos valores da percentagem para o total que foi de 83,05 % para o sistema A; 79,54% para o sistema B e de 42,92% para o sistema C.

Ainda relacionado com o problema de gerenciamento de espaço em sistemas de arquivos, é interessante ressaltar também a atenção e a abordagem tomada pelo Centro de Computação Eletrônica da USP com relação a arquivos não utilizados há mais de 5 meses pelos usuários do "mainframe" UNISYS A5. Neste caso os arquivos são copiados para fitas magnéticas e os usuários são informados do fato através de um "mail". A recuperação dos arquivos pelos usuários exige do usuário uma consulta a um arquivo diretório que tem a identificação da fita que foi usada para a cópia do arquivo. É necessário ainda que o usuário solicite ao operador a instalação da fita desejada para re-instalar o arquivo no disco magnético.

Pelas observações aqui expostas, fica clara a necessidade dos servidores de arquivos de possuírem mecanismos não só na área de segurança como também no que diz respeito ao gerenciamento de arquivos antigos (não acessados por muito tempo).

Antes de iniciar-se o detalhamento da proposta de trabalho, é apropriado considerar-se o sistema operacional que servirá de base para a implementação do sistema auxiliar de arquivos WORM bem como a importância dos servidores de arquivamento (archive servers).

5.3 SISTEMA OPERACIONAL MINIX

De acordo com Tanenbaum [TA87], o sistema UNIX está organizado como um programa executável simples, que é carregado para a memória durante a carga do sistema e então executado. O MINIX por sua vez está estruturado de uma maneira muito mais modular, como uma coletânea de processos que se comunicam entre si e com os processos dos usuários através de mensagens. Existem processos separados para gerenciamento de memória, para o sistema de arquivos, para cada "driver" de dispositivo do sistema e para certas funções. Cada um destes processos do sistema tem sua alocação própria de memória, tabela de processo e estado. Pode desse modo executar, aguardar, e enviar mensagens como um processo qualquer de usuário. O gerenciador de memória e o sistema de arquivos são executados individualmente no espaço de usuário. Já os "drivers" dos dispositivos são ligados em conjunto com o núcleo ("kernel") formando um único programa binário, que comunicam entre si e com outros processos via mensagens.

Quando o sistema é compilado, quatro programas binários independentes são criados: o núcleo (incluindo os "drivers"), o gerenciador de memória, o sistema de arquivos e init (que lê /etc/ttys e cria os processos "login"). Desse modo é possível recompilar uma única parte, ex. o sistema de arquivos, sem alterar as demais.

Os processos estão estruturados em quatro camadas:

4. Processos do usuário (camada alta)
3. Processos servidores (gerenciador de memória, sistema de arquivos)
2. "Drivers" de dispositivos (um processo por dispositivo)
1. Processos e manipuladores de mensagens (camada baixa)

A camada 1 ocupa-se com o gerenciamento de processos incluindo escalonamento da CPU e comunicação entre processos. Nesta camada são tratadas as interrupções, traps e mecanismos de passagem de mensagens.

A camada 2 contém os "drivers" que fazem parte do espaço do núcleo e são responsáveis pelos processos de entrada e saída. Para distinguir os processos dentro do núcleo daqueles do espaço do usuário, os processos do núcleo são chamados de tarefas.

A camada 3 contém somente dois processos, o gerenciador de memória e o sistema de arquivos. Ambos são estruturados como servidores que tem como clientes os processos de usuário. Cabe ao gerenciador de memória atender as

chamadas de sistema que necessitam de gerenciamento de memória (FORK, EXEC, BREAK, SIGNAL, etc). O sistema de arquivos responde pelas operações de arquivos (OPEN, READ, WRITE, etc). O sistema de arquivos projetado como servidor possibilita facilmente a substituição do sistema de arquivos local por um sistema remoto.

A camada 4 contém todos os processos dos usuários (editores, compiladores, shell, etc). Os processos no sistema MINIX podem criar subprocessos (filhos), que por sua vez podem criar mais subprocessos resultando em uma árvore de processos tendo como raiz o processo **init**. O processo **init** tem a função de criar processos filhos, um para cada terminal encontrado no arquivo */etc/ttys*. Cada processo filho executa o processo */bin/login* que fica aguardando que um usuário inicie o uso do terminal.

5.3.1 Sistema de Arquivos do MINIX

O sistema de arquivos do MINIX é um servidor remoto de arquivos real que roda no sistema do usuário. É possível portanto, convertê-lo diretamente em um verdadeiro servidor de arquivos de rede, sendo necessário apenas modificações na interface de mensagens e prover alguma forma de autenticação, conforme afirma Tanenbaum em [TA87].

Como no UNIX, o MINIX tem associado a cada arquivo uma pequena tabela (em disco) que é chamada de **i-node**. Esta tabela contém informações de proteção, proprietário, registros de criação, acesso e atualização e endereços dos blocos de dados do arquivo. Apesar do sistema de arquivos do MINIX ser similar ao do UNIX, o i-node é ligeiramente diferente, contendo apenas 9 endereços de disco contra 13 do UNIX e apenas 1 tempo (o da última modificação) em vez de 3 (criação, acesso e última modificação). O gerenciamento dos blocos e i-nodes livres é realizado em memória através de mapas de bits em vez de listas como no UNIX. O MINIX utiliza blocos e zonas de 1 K bytes e mantém um "cache" dos últimos blocos. A localização de um bloco no cache é feita através de um algoritmo de "hashing" e as alterações feitas em um bloco são tratadas da seguinte maneira: se o bloco for de i-nodes ou de diretório ou um bloco crítico, ele é imediatamente gravado; se for um bloco de dados, ele só será gravado através de um SYNC ou quando o cache for necessário para outros blocos.

A estrutura do sistema de arquivos do MINIX para qualquer dispositivo de armazenagem tem a forma apresentada na Fig. 5.1.

Fig. 5.1 - Estrutura do sistema de arquivos do MINIX para um dispositivo qualquer.

O **bloco de boot** existe em todos os sistemas de arquivos, mesmo em dispositivo que não é utilizado para carga do sistema operacional. Isto garante a uniformidade da estrutura dos sistemas de arquivos independentemente do dispositivo.

O **Super bloco** contém informações a respeito da estrutura do sistema de arquivos, fornecendo o tamanho do bloco, o número de i-nodes e o tamanho de cada parte do sistema de arquivos. A Figura 5.2, ilustra a estrutura do super bloco.

Fig. 5.2 - Super bloco do sistema de arquivos do MINIX.

O **mapa de bits do I-node** é um mapa de bits que permite gerenciar a criação e remoção de arquivos dos sistemas de arquivos. Para cada arquivo é associado um bit n designando o seu i-node n .

O **mapa de bits de Zona** tem uso semelhante ao mapa de bits do I-node, porém serve para gerenciar as zonas livres e ocupadas. Uma zona de um sistema de arquivos é constituída de vários blocos pertencentes a um mesmo cilindro, que proporciona melhor performance em acessos seqüenciais. Porém, a versão do MINIX em uso considera uma zona igual a um bloco que é igual a 1 k bytes.

A composição do i-node do MINIX é mostrada na Fig.5.3. O i-node do MINIX difere do UNIX em vários aspectos: 1) uso de apontadores de 2 bytes contra 3 do UNIX; 2) 9 apontadores contra 13; 3) uso de apenas um tempo, o que indica a última modificação do arquivo e 4) o número de ligações ao arquivo e o campo *gid* foram reduzidos a 1 byte. Estas alterações reduziram o tamanho do i-node de 64 para 32 bytes. Quando um arquivo é aberto, o seu i-node é localizado e trazido para a memória, onde é mantido até que o arquivo seja fechado.

Fig. 5.3 - Estrutura do i-node

A tabela de i-nodes é acrescida de informações que não se encontram no disco, tal como os números do i-node e do dispositivo onde se encontra o i-node. A principal função do i-node de um arquivo é dizer onde estão os blocos de dados do arquivo. Os endereços das primeiras sete zonas do arquivo são obtidos diretamente do i-node do arquivo. Desse modo, arquivos de até 7 k bytes não necessitam de blocos indiretos. Acima de 7 k, são necessárias zonas indiretas. Com blocos e zonas de 1 k e números de 16 bits, um bloco indireto contém 512 entradas, representando 512k de

armazenagem. Um bloco duplo indireto aponta para 512 blocos de indireto simples, totalizando 256 megabytes. Porém, devido ao uso de números de 16 bits e zonas de 1k, só é possível endereçar 64k zonas, ou 64 megabytes de espaço para armazenagem. Se for necessário um disco de maior capacidade é necessário aumentar o tamanho da zona, por exemplo 2k.

O sistema de arquivos do MINIX dispõe de um importante subsistema que gerencia diretórios e nomes de caminhos. O diretório do MINIX é constituído por um arquivo de entradas de 16 bytes, sendo 2 bytes usados para o número do i-node e 14 bytes para o nome do arquivo.

O gerenciamento dos arquivos abertos e em uso (READ, WRITE, etc) é realizado através de **descritores de arquivos**. Os descritores estão associados as tabelas de processos que dispõe de apontadores para os i-nodes do diretório raiz e do diretório corrente e um terceiro campo importante da tabela de processo é de um conjunto indexado pelo número do descritor.

Além da importância do sistema de arquivos do MINIX e do próprio MINIX para o desenvolvimento deste trabalho, considera-se relevante a abordagem proposta por Santana, em [SA90] a respeito de servidores de arquivamento (Archive Servers), que é apresentada a seguir.

5.4 SERVIDORES DE ARQUIVAMENTO (ARCHIVE SERVERS)

Santana, em [SA90], descreve servidores de arquivamento (Archive Servers), como servidores de arquivos com melhor capacidade de armazenagem e segurança. E propõe a utilização de disco ótico WORM para trabalhar em conjunto com dois discos magnéticos (Winchester). Justifica ainda, que isto aumenta a capacidade de armazenagem do sistema (da ordem de Gigabytes) e proporciona também maior segurança, uma vez que o disco ótico não pode ser apagado.

Para atingir os objetivos descritos, é proposto um servidor de arquivos organizado em três níveis lógicos:

- nível 1 - servidor corrente - sistema para armazenamento a curto prazo.
Composto pelos servidores de arquivos menos ocupados no momento em que os usuários acessam o sistema e onde são criados seus arquivos.

nível 2 - servidor padrão - sistema para armazenamento a médio prazo.
Servidor de arquivos padrão do usuário.

nível 3 - servidor de arquivamento a longo prazo - serviço de arquivamento dos "arquivos".

A transferência de arquivos de um nível de armazenamento para outro pode ser realizada através de dois mecanismos: "**Copy-Back**" e "**Archive**".

O mecanismo "**Copy-Back**" - Suporta a adoção de servidores correntes de arquivos e servidores padrão de arquivos. Este mecanismo transfere arquivos criados no nível 1 para o nível 2 e pode ter as seguintes abordagens:

a) Abordagem Centralizada

- uma lista atualizada de todos os diretórios com os arquivos a ser transferidos é mantida pelo servidor de diretório (esta lista é armazenada em disco para prevenir eventual falha ou "resets");
- todos os arquivos a ser transferidos para outros servidores de arquivos são mantidos em diretórios locais temporários;
- em um tempo pré-definido de ativação, o seguinte mecanismo de "copy-back" de arquivo é executado:

faça

percorre todos os diretórios selecionados e localiza todos os arquivos a serem transferidos;
organiza estes arquivos por usuário e por servidor de arquivo fonte;
para cada usuário faça:
 pega o servidor padrão de arquivo do usuário (a partir do servidor de nomes);
 para cada servidor de arquivo fonte faça:
 enquanto existir arquivo a transferir faça:
 seleciona um arquivo;
 abre um arquivo "output stream" com o servidor padrão de arquivo do usuário;
 requisita ao servidor de arquivos fonte a transferência do arquivo;
 atualiza a estrutura do diretório ao final da transferência;
 apaga o arquivo original do servidor de arquivos fonte;
 apaga o diretório local especial do servidor de arquivos fonte;
envia mensagem a todos os servidores de arquivos ativos informando que o mecanismo "copy-back" terminou;

fim.

b) Abordagem Distribuída

- todos os arquivos a serem transferidos para outro servidor de arquivo são mantidos nos diretórios locais temporários;

- em um tempo de ativação pré-definido o mecanismo "copy-back" de arquivo é executado:

faça

para cada diretório local temporário faça:
pega o servidor padrão de arquivos do usuário (a partir do servidor de nomes);
enquanto existir arquivo a transferir faça:
seleciona um arquivo;
abre o arquivo "output stream" com o servidor padrão de arquivos do usuário;
transfere o arquivo selecionado;
atualiza a estrutura do diretório ao final da transferência;
apaga o arquivo transferido do servidor de arquivos fonte;
apaga o diretório local temporário;
o termino do mecanismo de transferência no servidor de arquivos considerado
é informado a todos os outros servidores de arquivos ativos;

fim.

O mecanismo "**Archive**", é usado para transferir arquivos do nível 2 para o nível 3. Suporta a adoção de discos óticos e é usado para arquivar tanto arquivos com baixa probabilidade de serem atualizados como arquivos explicitamente indicados para arquivamento.

A abordagem básica para implementar o mecanismo de "archive" segue o que foi disposto para o mecanismo "copy-back". Contudo uma política de "archive" é necessária para selecionar quais arquivos serão incluídos no "archive".

Os arquivos com baixa probabilidade de atualização seriam fortes candidatos a serem transferidos para o WORM, uma vez que o número de versões (cópias) no WORM seria pequeno. Porém, poderia sobrecarregar o sistema devido a um aumento no número de acessos de leitura. A estratégia mais adequada é selecionar os arquivos para transferência por:

- **longevidade**: arquivos que há muito tempo não são acessados. Estes arquivos definem para o sistema auxiliar de arquivos o tipo **normal** de arquivo.
- **classe de usuário**: Uma política de arquivamento é implementada permitindo que arquivos sejam incluídos explicitamente em uma lista de arquivos a serem arquivados pelo gerente de sistema ou usuários autorizados.

A adoção de um servidor de arquivamento aumenta tanto a capacidade do sistema como sua confiabilidade. A política de arquivamento proposta por Santana

[SA90] tem potencial para permitir a utilização de discos WORM. A seguir é proposto um servidor auxiliar de arquivos WORM considerando diferentes requisitos tais como os dados obtidos nos sistemas analisados na seção 6.2.

5.5. CARACTERIZAÇÃO DO SISTEMA AUXILIAR DE ARQUIVOS WORM

O sistema auxiliar de arquivos WORM pode ser caracterizado pelos seguintes elementos básicos: arquivos envolvidos, mecanismo de transferência e o disco WORM.

5.5.1 Arquivos envolvidos no sistema auxiliar de arquivos WORM

Considerando-se aspectos de segurança e disponibilidade de espaço do servidor de arquivos, serão considerados três tipos de arquivos para gravação em WORM: **crítico, normal e morto**:

Crítico - Um arquivo é considerado crítico quando não se admite perda de informações ou da disponibilidade do arquivo. Neste caso o servidor auxiliar de arquivos WORM proverá ao sistema de arquivos uma cópia adicional em WORM, toda vez que o arquivo sofrer alteração.

Normal - Qualquer arquivo gerenciado pelo servidor de arquivos.

Morto - possibilidade de adotar-se cópias em WORM de algumas classes especiais de arquivos em processo de remoção do sistema de arquivos, tais como: fontes, dados, arquivos de usuários especiais, etc.

Para atender a esta classificação dos arquivos é necessário criar uma categoria de usuário especial do sistema com privilégios (habilitado) para definir arquivos críticos e permitir copiar a qualquer instante um arquivo qualquer para o WORM.

A gravação e recuperação de cópias de arquivos do WORM dependem de mecanismos de gerenciamento de transferência entre o meio magnético e o WORM. Em função da classificação adotada para os arquivos em crítico, normal e

morto, estes mecanismos serão diferenciados e apresentados para cada classe de arquivos.

5.5.2 Mecanismos de transferência de arquivos no sistema auxiliar WORM

O mecanismo adotado para transferência dos arquivos será uma adaptação do mecanismo "Archive", conforme proposto por [SA90] e descrito anteriormente. A seguir, apresenta-se uma descrição do mecanismo adequado a cada categoria de arquivo do sistema auxiliar de arquivos WORM.

Arquivo Crítico. A transferência de arquivo crítico para o disco WORM pode ser feita a qualquer instante pelo servidor auxiliar de arquivos WORM. Toda vez que um arquivo é definido como crítico ou um arquivo crítico é alterado é necessário que o servidor auxiliar de arquivos WORM faça imediatamente uma nova cópia do arquivo, para ter assegurado no menor tempo possível a integridade das informações. Uma lista atualizada de todos os diretórios com os arquivos a serem transferidos é mantido pelo servidor WORM e a transferência dos arquivos críticos para o WORM é possível graças à execução periódica, do seguinte mecanismo de "archive":

```
repita
  Recebe do servidor de arquivos a identificação do último arquivo alterado
  "id_arq";
  Se id_arq consta da lista de arquivos críticos
    então
      faz nova cópia (versão) do arquivo no WORM;
      atualiza diretório WORM;
    fimse
  fim
```

Para que o processo acima tenha êxito, é necessário que haja um procedimento que permita ao usuário habilitado, indicar seus arquivos críticos. Uma alternativa pode ser dada por:

```
faça
  Recebe do usuário habilitado identificação de arquivo crítico;
  Atualiza lista de arquivos críticos do WORM;
end
```

Arquivo Normal. A transferência de um arquivo normal para o disco WORM não exige a mesma urgência dos arquivos críticos. Desse modo é possível escalar a transferência de um arquivo normal sempre que o sistema apresentar ociosidade. A transferência pode ocorrer por 2 motivos:

a. **Longevidade** (ou desuso). A permanência dos arquivos normais (conteúdo) no disco magnético é assegurada por um período adequado de n_dias não acessados. O tempo de permanência dos arquivos (n_dias) pode ser atualizado estatisticamente pelo servidor WORM tornando-se mais eficiente em relação à carga de transferência do meio magnético para o WORM e vice-versa. Todo arquivo que estiver com seu tempo de permanência no disco magnético vencido, será transferido para o disco WORM. O conteúdo do arquivo será removido do disco magnético e um "link" para o diretório de uso do WORM será criado. Assim, quando for necessário, será possível a re-instalação de uma nova cópia no meio magnético. Neste caso o mecanismo de transferência é o seguinte:

faça

percorre todos os diretórios e localiza os arquivos a serem transferidos;

enquanto existir arquivo a transferir faça

cria "link" para o arquivo no diretório de uso do WORM;

copia arquivo para o WORM;

remove conteúdo do arquivo do disco magnético;

atualiza diretório WORM;

fimenq

fim.

b. **Solicitação do usuário.** O usuário habilitado poderá a seu critério solicitar a transferência de qualquer um de seus arquivos para o WORM, bastando para isso informar ao sistema auxiliar de arquivos WORM a identificação do seu arquivo. Neste caso o sistema auxiliar de arquivos WORM acrescentará na lista de arquivos a transferir o arquivo solicitado pelo usuário para futura transferência para o disco WORM. O conteúdo do arquivo original permanece no disco magnético. Neste caso, tem-se uma duplicata do arquivo no WORM.

faça

percorre a lista de arquivos a serem transferidos;

enquanto existir arquivo a transferir faça

copia arquivo para o WORM;

atualiza diretório WORM;

fimenq

fim

Do mesmo modo como para os arquivos críticos, é necessário que um procedimento permita ao usuário habilitado, informar qual arquivo deseja copiar para o WORM. Neste caso o procedimento pode ser:

faça

Recebe do usuário habilitado identificação do arquivo normal;

Atualiza a lista de arquivos normais do WORM;

fim.

Arquivo Morto. A viabilidade deste modo de transferência depende do processo de remoção definitiva de arquivo do meio magnético. Normalmente o procedimento de remoção de arquivo de um sistema de arquivos dispõe de parâmetro de controle para evitar a remoção acidental. Como exemplo no MINIX (UNIX) o comando: `rm -i arquivo`, interroga o usuário se realmente deseja remover o arquivo. Neste caso é possível cancelar a tempo uma remoção acidental. Porém, se o parâmetro `-i` não for especificado e o usuário tiver permissão para escrever no arquivo, a remoção será concluída sem possibilidade de reaver o arquivo. O que se pretende é propor um novo processo de remoção que permita transferir para o WORM algumas classes especiais de arquivos: fontes, dados, de usuários particulares, etc. Desse modo, o WORM funcionaria também como uma espécie de coletor de lixo, executando o procedimento a seguir, sempre que a remoção de arquivos especiais fossem solicitadas:

faça

percorre a lista de arquivos a serem transferidos;

enquanto existir arquivo a transferir faça

 copia arquivo para o WORM;

 remove arquivo do meio magnético;

 atualiza diretório WORM;

fimenq

fim.

O sistema auxiliar de arquivos WORM dispõe de um escalador de transferência que gerencia continuamente as listas de arquivos críticos e normais, indicando sempre que existir qual o próximo arquivo a ser transferido para o WORM pelo processo copiador.

Para agilizar a transferência e garantir uma cópia o mais rápido possível para os arquivos críticos e normais solicitados por usuário, o sistema auxiliar de arquivos WORM, fará uma duplicata do arquivo no próprio disco magnético em um diretório de uso exclusivo do WORM. Com esta abordagem se consegue a mesma função de "cache", apesar de poder sobrecarregar o disco magnético dependendo da demanda de transferência para o disco WORM. Assim que a duplicata é transferida para o WORM, ela é removida do disco magnético.

No caso de arquivos que estão sendo removidos por longevidade não há necessidade da duplicata, em vez disso, é lançado no diretório de uso do WORM um "link" para o arquivo. Na verdade, se tem também uma duplicata porém, só da identificação do arquivo e não do seu conteúdo como nos casos anteriores. Quando

este arquivo é copiado para o WORM, o seu conteúdo é removido mas é mantida a sua entrada no diretório original.

A necessidade de re-instalar, no disco magnético, uma determinada cópia de arquivo que está no WORM pode se dar de duas maneiras:

1. Quando um processo necessita de um arquivo removido para o WORM por longevidade, a re-instalação será automática. Isto é possível porque o sistema de arquivos, através de uma nova operação OPEN, é capaz de reconhecer este tipo de arquivo pelo número de ligações ao arquivo, tamanho do arquivo (zero) e finalmente indicação do diretório de arquivos do WORM. Uma vez reconhecido o sistema de arquivos solicita ao sistema auxiliar de arquivos WORM a imediata re-instalação da cópia do arquivo no disco magnético. Todo este processo ocorre durante a operação OPEN, sendo desse modo transparente para o usuário.
2. Uma cópia de uma versão qualquer de um arquivo transferido para o WORM pode ser solicitada pelo usuário habilitado ao sistema auxiliar de arquivos WORM em qualquer instante.

É importante ressaltar que o usuário comum não precisa conhecer os mecanismos de acesso ao disco ótico, que são totalmente transparentes. Somente usuários (habilitados) que necessitam de armazenamento no disco ótico devem conhecer os mecanismos para restaurar eventuais cópias que estejam no WORM.

A organização do sistema de arquivos do disco WORM proposto independe do sistema operacional e é apresentada a seguir.

5.5.3 Estrutura do sistema de arquivos do disco WORM

Uma vez que o disco WORM possibilita acesso aleatório e considerando-se a finalidade da escolha do mesmo como medida de segurança de arquivos, o disco será dividido em três regiões: de arquivos, livre e de diretório. Estas regiões não têm limites pré-estabelecidos a não ser pela capacidade de armazenamento do disco WORM. A Figura 5.4 ilustra a estrutura do disco WORM.

Fig. 5.4 - Estrutura do sistema de arquivos do disco WORM.

A região de arquivos, criada a partir do início do disco, é constituída por agrupamento de blocos gravados correspondentes aos arquivos. Cada agrupamento, que corresponde a um arquivo, é constituído de um bloco de cabeçalho de identificação do arquivo seguido pelos blocos de dados do arquivo. Desse modo cada arquivo gravado no WORM pode ser representado pelas seguintes estruturas:

Arquivo:

```
struct WORM_arq {
    cabeçalho id_arquivo;
    conteúdo dados_arquivo;
}
```

Sendo:

cabeçalho:

```
entr_dir cabeçalho
```

conteúdo (dados):

```
struct conteúdo {
    char dados_arq[TAMANHO_ARQUIVO];
}
```

A segunda região, livre ou intermediária, constitui-se dos blocos livres, que permitem a gravação de blocos da região de arquivos ou da região de diretório. A medida que as regiões 1 e 3 crescem, esta região diminui na mesma proporção.

A terceira região, de diretório, criada a partir do final do disco, é constituída pelos blocos gravados do diretório. A escolha de uma região para diretório se deve a dois motivos: primeiro, maior rapidez na localização de um arquivo e reconstituição do diretório em disco magnético; segundo, replicação por segurança das entradas dos arquivos no WORM. Considerando-se que todo arquivo gravado no

WORM é constituído de um cabeçalho seguido dos blocos de dados, não haveria necessidade de uma região dedicada a diretório, conseqüentemente, se aumentaria a área disponível para a região 1. Porém, a localização de um arquivo no WORM, utilizando-se somente a região 1, exige a leitura de muitos blocos. Considere-se, por exemplo, a necessidade de localizar um arquivo que tenha sido a entrada número 1000 no WORM. Neste caso é necessária a leitura de 1000 blocos correspondentes as 999 entradas (cabeçalhos) dos arquivos anteriores mais uma leitura da entrada do arquivo em questão. Este mesmo caso, considerando-se a utilização do diretório, exige um número bem menor de leituras do WORM que depende do tamanho do bloco e do tamanho de cada entrada de arquivo no diretório. Supondo-se que cada bloco do WORM tenha 512 bytes e que cada entrada tenha 128 bytes, portanto 4 entradas de arquivos por bloco, a redução neste caso é de 4 acessos da região 1 para 1 acesso da região 3. Logo, a entrada do arquivo número 1000 é localizada com a leitura de apenas 250 blocos da região 3.

Outra consequência da utilização da região 3 que determina a rapidez da recuperação do arquivo é que o acesso na região 3 se dá sequencialmente enquanto que o acesso da região 1 é aleatório. Com base nestas considerações poderia se admitir que o cabeçalho não desempenha nenhuma função de importância. Mas, como segurança é uma questão relevante para o saa-WORM, a necessidade do cabeçalho e do diretório se justifica. Considerando-se que cada bloco do diretório seja gravado no WORM somente quando este estiver completo (por exemplo: com 4 entradas de arquivos), uma eventual falha durante a gravação de um bloco do diretório perderia 4 entradas de arquivos, e como consequência a impossibilidade de recuperação dos 4 arquivos correspondentes. Com a replicação da entrada do arquivo através do cabeçalho torna-se possível não só a recuperação dos arquivos como a possibilidade de restaurar o diretório.

O diretório do WORM faz parte das estruturas que o sistema auxiliar de arquivos WORM necessita ter em disco magnético com o propósito de melhorar o desempenho do sistema. A utilização do disco magnético como apoio ao saa-WORM se deve à grande diferença do tempo de acesso do disco magnético (10 ms) contra o WORM (85 ms), além do que, as informações dos arquivos existentes no WORM são geradas a partir do disco magnético.

A estrutura proposta para cada entrada no diretório do WORM como para cada entrada do arquivo de diretório geral mantido no disco magnético, é a seguinte:

```

struct entr_dir {
    short dev                /* Número do dispositivo do arquivo */
    unsigned short inode_dir /* I-node do diretório que contém o arquivo */
    unsigned short inode_arq /* I-node do arquivo */
    long tam_arq             /* Tamanho do arquivo (bytes) */
    long d_versao            /* Data/hora da versão */
    int nmidia               /* Número da mídia que contém a cópia do arquivo */
    long bloc_worm           /* Endereço bloco início da cópia no WORM */
    int uid                  /* Número do usuário */
    int gid                  /* Número do grupo do usuário */
    char chave[6]            /* Chave de confirmação do cabeçalho */
    char nome_abs[92]        /* Nome absoluto do arquivo */
}

```

Com esta estrutura é possível o gerenciamento de várias unidades simultâneas de discos óticos no sistema auxiliar de arquivos WORM e a correta ligação com a mídia WORM utilizada na gravação do arquivo, mesmo quando removida do sistema. O saa-WORM controla a instalação e remoção de mídia WORM através de uma estrutura mantida em disco magnético contendo informações do dispositivo onde está instalado a mídia WORM e a identificação (número) da mídia.

5.6. COMENTÁRIOS

Neste capítulo foi dado ênfase à necessidade dos servidores de arquivos de possuírem mecanismos não só na área de segurança como também no que diz respeito ao gerenciamento de espaço disponível, com atenção especial aos arquivos com baixa probabilidade de acesso.

Mostrou-se através de levantamento de alguns sistemas de arquivos o comprometimento de espaço em disco por arquivos não acessados a mais de 180 dias. Apesar do avanço tecnológico proporcionar discos magnéticos cada vez mais poderosos em capacidade e tempo de acesso, cresce também o número de usuários, consequentemente aumenta a demanda por espaço. Como pôde ser observado nos sistemas analisados, a área em uso corresponde a 86%, 76% e 78%, para os sistemas A, B e C respectivamente. Sendo que a área comprometida com arquivos não acessados a mais de 180 dias corresponde a 83%, 79% e 43%, da área em uso, respectivamente aos sistemas A, B e C. Os sistemas A e B de menor capacidade de armazenamento são os mais comprometidos com os arquivos que não são acessados a mais de 180 dias. Mesmo o sistema C tem um comprometimento expressivo que é

de 33% da área total do sistema de arquivos. Considerando-se a importância da liberação desta área (33%) e a média de área em uso por usuário (435.727/715) que é de 305 kbytes, pode-se estimar um acréscimo de 307 usuários para o sistema de arquivos C. O número de usuários que é possível aumentar corresponde exatamente ao percentual da área comprometida pela área em uso, assim os sistemas A, B e C podem ter o número total de usuários acrescido em 83%, 79% e 43%, respectivamente.

Com o propósito de subsidiar o detalhamento da proposta do saa-WORM, apresentou-se características gerais do sistema operacional MINIX, com atenção especial para o sistema de arquivos. E ressaltou-se também os mecanismos de gerenciamento de transferência de arquivos entre servidores de arquivos apresentados por Santana [SA90].

A proposta do servidor auxiliar de arquivos para disco ótico WORM foi apresentada como solução alternativa para os problemas apresentados de segurança e gerenciamento de arquivos de baixa probabilidade de acesso.

Considerando-se os aspectos de segurança e disponibilidade de espaço do servidor de arquivos, três tipos de arquivos para gravação em WORM foram propostos: **crítico**, **normal** e **morto**. Sendo que um arquivo é:

Crítico - Quando não se admite perda de informações ou da disponibilidade do arquivo (a critério do usuário). Neste caso o saa-WORM proverá ao sistema de arquivos uma cópia adicional em WORM, toda vez que o arquivo sofrer alteração.

Normal - Para qualquer arquivo gerenciado pelo servidor de arquivos.

Morto - Quando há interesse de adotar-se cópias em WORM de algumas classes especiais de arquivos em processo de remoção do sistema de arquivos, tais como: fontes, dados, arquivos de usuários especiais, etc.

Esta classificação evidencia a importância da cópia em WORM como medida de segurança - replicação do arquivo em dispositivo mais robusto. Quanto a disponibilidade de espaço do servidor de arquivos, foi proposto para os arquivos normais um mecanismo de remoção para o WORM baseado na longevidade dos arquivos, isto é, o tempo de permanência dos arquivos nos discos magnéticos sem acesso. Considerou-se como fator determinante para a remoção dos arquivos normais para o WORM o tempo de 90 dias. Assim todo arquivo normal com mais de 90 dias é removido do disco magnético para o WORM. Esta remoção pode significar um

ganho de espaço importante para o servidor de arquivos, como pode ser visto pelos sistemas de computação analisados na seção 6.2, onde estes arquivos comprotem entre 43% a 83% da área em uso.

A remoção dos arquivos normais por longevidade não impõe nenhum obstáculo a futuros acessos a estes arquivos. Contudo, pode haver um aumento do tempo de resposta em decorrência da necessidade de re-instalar o arquivo no disco magnético. O saa-WORM mantém a entrada original do arquivo no disco magnético e cria um "link" ao arquivo para permitir a re-instalação do arquivo quando for necessário. Este processo de remoção do arquivo normal (longevidade) para o WORM e re-instalação no disco magnético é totalmente transparente para as duas categorias de usuários, habilitados e não habilitados. A manipulação dos arquivos críticos, uma vez definidos pelos usuários habilitados, é também transparente a todos os usuários e ocorre sempre após uma alteração.

Para o saa-WORM atender a transferência de arquivos críticos e arquivos normais foi proposto a divisão dos usuários em duas categorias: usuários habilitados (com permissão para usar o saa-WORM) e usuários não habilitados. Deste modo fica a critério do usuário habilitado indicar seus arquivos críticos ao saa-WORM e solicitar quando necessário cópia de um arquivo normal para o WORM.

A estrutura do sistema de arquivos proposta para o disco WORM considerou o disco dividido em três regiões: de arquivos, livre e de diretório. Estas regiões não têm limites pré-estabelecidos a não ser pela capacidade de armazenamento do disco WORM.

A proposta do sistema auxiliar de arquivos WORM foi realizada considerando-se aspectos importantes para o usuário como a transparência e a independência do sistema de arquivos.

IMPLEMENTAÇÃO DO SISTEMA AUXILIAR DE ARQUIVOS WORM

6.1 INTRODUÇÃO

A implementação foi escolhida como forma de demonstrar a viabilidade do sistema proposto. Na impossibilidade da aquisição de uma unidade de disco ótico WORM, utilizou-se a unidade de disquete para implementação de todos os mecanismos definidos para o sistema auxiliar de arquivos WORM. Apesar da capacidade de armazenagem ser muito menor do que a mídia WORM, o disquete oferece condições satisfatórias para simulação do sistema proposto.

A implementação deste sistema foi realizada em linguagem de programação C, através de alterações de procedimentos (OPEN, CLOSE) do sistema de arquivos do MINIX bem como do desenvolvimento de vários programas de suporte e gerenciamento de arquivos transferidos do meio magnético para o WORM e vice-versa.

A abordagem básica da implementação constitui-se de um processo, chamado de **wgt** - gerenciador de transferência, disparado na carga do sistema operacional MINIX através da execução pelo "shell" do arquivo `/etc/rc`. Este processo instala-se por si só em "background" e tem a função de gerenciar as listas de arquivos a serem copiados para o WORM.

As listas de arquivos foram implementadas como subdiretórios de um diretório criado em disco magnético para uso exclusivo do WORM. Para os arquivos críticos, foi criado o subdiretório **crítico**, para os arquivos a serem removidos por

longevidade criou-se o subdiretório **desuso** e finalmente para os arquivos a serem copiados por solicitação de usuários, foi criado o subdiretório **copia**. Esta abordagem permitiu com facilidade e segurança, implementar um processo de cópia rápida, ou seja, uma duplicata do arquivo no próprio disco magnético. O gerenciamento destas listas (subdiretórios) foi realizado com a utilização de um arquivo diretório, **worm.dir**, contendo todas as entradas (identificações dos arquivos) das listas. O detalhamento individual dos componentes do sistema proposto será apresentado neste capítulo.

6.2. MÓDULOS COMPONENTES DO SISTEMA AUXILIAR DE ARQUIVOS WORM

Antes de considerar os componentes do sistema auxiliar de arquivos WORM, é interessante ressaltar as alterações que foram necessárias nos procedimentos OPEN e CLOSE do sistema de arquivos do MINIX. Inicialmente será abordado o procedimento CLOSE porque relaciona-se diretamente com os arquivos críticos. Como já foi mencionado, os arquivos críticos devem ser copiados para o WORM imediatamente após alteração. Uma forma possível para a realização desta tarefa é a indicação do sistema de arquivos do MINIX de que um arquivo crítico foi alterado. E o melhor momento para isso é o fechamento do arquivo, ou seja, durante a execução do CLOSE.

O procedimento CLOSE foi então alterado para que toda vez que um arquivo for fechado, seja feita uma consulta a lista de arquivos críticos e em caso afirmativo a indicação ao sistema auxiliar de arquivos WORM é dada pelo disparo de um processo copiador (cp do MINIX). O copiador então encarrega-se de fazer uma cópia do arquivo crítico no próprio disco magnético em um subdiretório, denominado crítico, de uso do WORM. Esta cópia permanece no subdiretório crítico até que seja definitivamente copiada para o disco WORM, sendo então removido o conteúdo do arquivo.

A necessidade de alteração do OPEN se deve aos arquivos removidos para o WORM por longevidade. Toda vez que um destes arquivos é necessário a um processo qualquer não é possível ao sistema de arquivos do MINIX atender imediatamente, uma vez que os conteúdos destes arquivos não estão disponíveis. Assim o momento adequado para notificar ao sistema auxiliar de arquivos WORM ocorre na execução do OPEN. Neste caso, toda vez que um arquivo é aberto, a lista de arquivos

removidos por longevidade é consultada e se for o caso, então o OPEN indica ao sistema auxiliar de arquivos WORM que é necessário re-instalar a última cópia do arquivo no disco magnético. Isto é feito disparando-se um processo re-instalador de cópias do WORM para disco magnético (**wrc**).

No Apêndice A são apresentados os códigos originais e alterados correspondentes ao OPEN e ao CLOSE.

Além das alterações em procedimentos do sistema de arquivos do MINIX, a implementação proposta necessitou de novos procedimentos constituídos dos seguintes módulos básicos:

- wuser** - Autorização de usuários,
- wgt** - Gerenciador de transferência de arquivos para WORM,
- wrc** - Re-instalador de cópias de arquivos do WORM para o meio magnético,
- wdes** - Coletor de arquivos que não são acessados há mais de *n-dias*,
- wci** - Coletor de arquivos críticos,
- wcp** - Coletor de arquivos normais solicitados por usuário.

O processo autorização (habilitação) de usuário (**wuser**) permite ao supervisor cadastrar usuários habilitados a usarem o sistema auxiliar de arquivos WORM, dividindo-se assim, os usuários em duas categorias. A primeira categoria é formada por usuários habilitados para utilizar o sistema auxiliar de arquivos WORM explicitamente, seja definir arquivos críticos, ou fazer cópia de arquivos normais. A segunda categoria é formada pelos usuários que terão arquivos normais (longevidade) armazenados no disco WORM de forma transparente. Os usuários habilitados são mantidos pelo saa-WORM em um arquivo de usuários (**wusers**) no diretório worm.

O Gerenciador de transferência de arquivos para WORM (**wgt**) é a interface principal entre o sistema de arquivos do MINIX e o disco WORM (simulado pelo disquete). O wgt é disparado na carga do sistema operacional MINIX através da execução pelo "shell" do arquivo **/etc/rc**. Uma vez ativado, ele roda continuamente em "background" gerenciando as listas de arquivos a serem copiados para o WORM. Estas listas, que são os subdiretórios crítico, cópia e desuso, são percorridos continuamente nesta ordem a procura de arquivos a serem transferidos para o disco WORM (disquete). Existindo arquivo para transferir o wgt ativa um processo copiador (neste caso um subprocesso do próprio wgt) que faz a transferência do arquivo do disco magnético para o WORM (disquete). Como pode-se ver a prioridade é sempre dada

à lista de arquivos críticos, seguida dos arquivos solicitados por usuário e finalmente aos arquivos normais a serem removidos por longevidade.

Além de ser responsável pela transferência dos arquivos contidos nos subdiretórios, o **wgt** se encarrega de atualizar o arquivo de diretório do WORM (**worm.dir**), bem como remover os arquivos não mais necessários dos subdiretórios correspondentes, mantendo as listas ou subdiretórios atualizados.

O procedimento re-instalador de cópias de arquivos do WORM para o meio magnético - **wrc**, consiste somente da tarefa de copiar uma versão de um arquivo solicitado para o disco magnético e é solicitado em duas circunstâncias: 1) Quando um processo necessita de um arquivo removido para o WORM por longevidade, e 2) Quando um usuário habilitado solicita uma cópia de uma versão qualquer de um arquivo transferido para o WORM.

O Coletor de arquivos que não são acessados há mais de *n-dias* (90) - **wdes**, é ativado uma única vez durante o dia, por exemplo, durante a carga do sistema operacional ou programado através do cron para executar em uma hora de menor atividade do sistema. O **wdes** tem a função de vasculhar todos os diretórios de um sistema de arquivos localizando e preparando a lista dos arquivos não acessados a mais de 90 dias.

O objetivo do Coletor de arquivos críticos - **wci**, é proporcionar ao usuário habilitado indicar seu arquivo crítico e indicar ao saa-WORM através da ativação do processo copiador (**cp**) que existe um novo arquivo crítico na lista. Deste modo toda vez que um novo arquivo crítico é definido, uma cópia deste arquivo é feita no WORM (disquete).

O Coletor de arquivos normais solicitados por usuário - **wcp** é utilizado toda vez que um usuário habilitado requisite uma cópia de um arquivo qualquer armazenada no disco ótico. Neste caso o **wcp** acrescenta na lista de arquivos normais a solicitação do usuário através da ativação do processo **cp**.

Além dos procedimentos apresentados, outros foram elaborados a fim de auxiliar a implementação do sistema saa-WORM. São eles: listagem do diretório da mídia em uso (**wlw**), listagem do diretório geral do WORM em disco magnético (**wld**), inicializador de mídia (**wim**), montador de mídia (**wmm**), e desmontador de mídia (**wdm**) de um dispositivo WORM (disquete).

Para se obter do disquete o mesmo efeito de uma mídia WORM, foi necessário desenvolver um procedimento de inicialização de disquete. Este procedimento permite inicializar todos os blocos do disquete com um padrão de byte arbitrário, no caso igual a 255. Uma vez inicializado é possível identificar as três áreas distintas existentes em cada mídia, ou seja: 1) área de arquivos, 2) área livre e 3) área do diretório.

As listagens correspondentes aos códigos em linguagem C dos procedimentos apresentados encontra-se no Apêndice B.

A instalação do sistema auxiliar de arquivos WORM no sistema MINIX necessita não só dos procedimentos aqui apresentados individualmente, mas de uma preparação ordenada e bem definida dos procedimentos e estruturas envolvidas com o sistema. Este tópico é detalhado nas próximas seções.

6.3 ESTRUTURA INICIAL

A fim de facilitar a instalação do sistema auxiliar de arquivos WORM no sistema MINIX criou-se um arquivo de comandos ("shell") (**winstala.sh**) em disquete que prepara todas as estruturas necessárias a utilização do saa-WORM em disco magnético. O sistema de arquivos **/usr** do MINIX foi escolhido por ser o sistema de arquivos principal e estar montado em uma partição de disco magnético de 20 megabytes. A seguir, é apresentado o procedimento **wintala.sh** com todas as etapas (instruções) necessárias para a criação das estruturas: diretório de uso do WORM, subdiretórios (listas) de arquivos críticos, cópia e desuso, arquivo de cadastro de usuários e arquivo de diretório do WORM. Este procedimento deve ser executado pelo super-usuário.

```
cd /usr                                # Selecionar diretório /usr
mkdir worm                            # Criar sub-diretório de uso do WORM:
cd worm                                # Selecionar sub-diretório worm:
# Criar sub-diretórios correspondentes as listas dos arquivos: crítico,
# normal e longevidade:
mkdir crítico
mkdir copia
mkdir desuso
```

wuser root

habilita super-usuário

Em seguida, o arquivo `/etc/rc` do MINIX deve ser acrescido dos seguintes comandos:

```
/usr/worm/wgt          # Gerenciador de Transferência do WORM  
/usr/worm/wdes        #Coletor de arquivos normais - longevidade
```

Terminada a instalação é necessário ativar o saa-WORM, sendo conveniente re-carregar o sistema MINIX.

6.4 PROCEDIMENTOS TRANSPARENTES AO USUÁRIO

O saa-WORM dispõe de procedimentos que são totalmente transparentes ao usuário. São eles o **wgt**, **wdes**, **open** e **close**.

O **wgt**, como já foi definido, é o gerenciador de transferência de arquivos para WORM sendo a interface principal entre o sistema de arquivos do MINIX e o disco WORM (simulado pelo disquete). O **wgt** é disparado na carga do sistema operacional MINIX através da execução, pelo "shell", do arquivo `/etc/rc`. Uma vez ativado, ele roda continuamente em "background" gerenciando as listas de arquivos a serem copiados para o WORM, subdiretórios, crítico, cópia e desuso. Estes diretórios são percorridos continuamente nesta ordem a procura de arquivos a serem transferidos para o disco WORM (disquete). Existindo arquivo para transferir o **wgt** ativa um processo copiador (neste caso um subprocesso do próprio **wgt**) que faz a transferência do arquivo do disco magnético para o WORM (disquete). O pseudo código para o **wgt** é o seguinte:

```
repete  
  pega uma lista de (crítico, copia, desuso)  
  percorre a lista de arquivos a serem transferidos;  
  enquanto existir arquivo a transferir faça  
    copia arquivo para o disco WORM (disquete);  
    remove conteúdo do arquivo do meio magnético;  
    atualiza entrada da lista e do diretório WORM;  
  fimenq  
  próxima lista  
fim.
```

O **wdes** que tem como função coletar arquivos que não são acessados há mais de *n-dias* (90), é ativado uma única vez durante o dia, por exemplo, durante a carga do sistema operacional ou programado através do cron para executar em uma hora de menor atividade do sistema. Quando ativado o **wdes** percorre todos os diretórios do sistema de arquivos /usr localizando e preparando a lista dos arquivos não acessados a mais de 90 dias. As funções básicas do **wdes** podem ser apresentadas através do seguinte pseudo código:

faça

percorre /usr e sub-diretórios e localiza todos os arquivos a serem transferidos
(não acessados a mais de 90 dias);

enquanto existir arquivo a transferir faça

cria "link" para o arquivo no sub-diretório /usr/worm/desuso;

atualiza diretório WORM (worm.dir);

fimenq

fim.

O procedimento **open** da biblioteca do MINIX foi alterado para que toda vez que um arquivo for aberto, a lista de arquivos removidos por longevidade seja consultada e se for o caso, então o OPEN indica ao saa-WORM que é necessário re-instalar a última cópia do arquivo no disco magnético. Isto é feito disparando-se um processo re-instalador de cópias do WORM para disco magnético (**wrc**). O pseudo código para o **open** é o seguinte:

faça

se o arquivo tem característica de arquivo removido por longevidade, isto é:
número de links maior que 1 e tamanho zero, então:

se encontrar o arquivo no /usr/worm/desuso, então:

ativar re-instalação do arquivo do disco WORM (disquete) para o disco
magnético através do **wrc**

fimse

fimse

abre o arquivo para o usuário

fim

O procedimento **close** da biblioteca do MINIX foi alterado para que toda vez que um arquivo for fechado, uma consulta a lista de arquivos críticos seja feita e em caso afirmativo a indicação ao saa-WORM seja dada pela ativação do processo copiador (**cp** do MINIX). O copiador encarrega-se de fazer uma cópia do arquivo crítico no próprio disco magnético no subdiretório /usr/worm/crítico. Esta cópia permanece no subdiretório crítico até que seja definitivamente copiada para o disco

WORM (disquete), sendo então removido o conteúdo do arquivo. O pseudo código correspondente é o seguinte:

```
faça
  se encontrar o arquivo no /usr/worm/crítico, então
    se a data da última alteração do arquivo é mais recente que a do disco
      WORM, então
        ativa o copiador (cp) fazendo uma cópia no /usr/worm/crítico
    fimse
  fimse
  fecha o arquivo
fim
```

Uma vez que os procedimentos open e close foram alterados é necessário que todos os procedimentos que utilizem arquivos envolvidos com o saa-WORM sejam recompilados.

Os arquivos envolvidos nesta seção foram arquivos do tipo crítico e normal não acessados a mais de 90 dias (longevidade). Para atender aos outros arquivos do tipo normal há necessidade da intervenção do usuário. Em decorrência disto o saa-WORM dispõe de comandos que permitem ao usuário habilitado definir arquivos críticos, copiar qualquer arquivo para o disco WORM (disquete) e restaurar qualquer cópia de arquivo do disco WORM (disquete) para o disco magnético.

6.5 COMANDOS DISPONÍVEIS AO USUÁRIO

Todo usuário habilitado a usar o saa-WORM tem a sua disposição os seguintes comandos: **wci**, **wcp** e **wrc**.

O comando **wci** permite ao usuário definir seu arquivo crítico e tem a seguinte sintaxe: **wci** <identificação do arquivo crítico>. A função do **wci** é indicar ao saa-WORM através da ativação do copiador (cp) o novo arquivo crítico. O pseudo código correspondente é o seguinte:

```
faça
  se usuário habilitado, então:
    lê identificação do arquivo crítico id_arq
    ativa o copiador (cp) fazendo uma cópia de id_arq em /usr/worm/crítico
  fimse
fim
```

Através do comando **wcp** é possível ao usuário copiar qualquer arquivo do disco magnético para o disco WORM (disquete). Semelhante ao **wci**, tem a seguinte sintaxe: **wcp <identificação do arquivo>**. A função do **wcp** é indicar ao saa-WORM através da ativação do copiador (**cp**) que existe um novo arquivo a ser transferido para o disco WORM (disquete). O pseudo código é o seguinte:

```
faça
  se usuário habilitado, então:
    lê identificação do arquivo id_arq
    ativa o copiador (cp) fazendo uma cópia de id_arq em /usr/worm/copia
  fimse
fim
```

O comando **wrc** permite ao usuário re-instalar uma cópia do arquivo do WORM (disquete) para o disco magnético. A sintaxe para o comando **wrc** é a seguinte: **wrc <identificação do arquivo>**. O pseudo código correspondente é o seguinte:

```
faça
  se usuário habilitado, então:
    lê identificação do arquivo id_arq
    se id_arq consta do diretório WORM (worm.dir), então:
      ativa o wrc trazendo uma cópia de id_arq do disco WORM disquete para
      o disco magnético;
    fimse
  fimse
fim
```

A seguir são apresentados alguns comandos adicionais de apoio ao usuário. Estes comandos permitem ao usuário, obter informações sobre os arquivos existentes no disco WORM e ao super-usuário gerenciar o saa-WORM.

6.6 OUTROS COMANDOS DE APOIO AO USUÁRIO

Os comandos de apoio disponíveis ao usuário são divididos em dois grupos. O primeiro grupo dispõe de comandos de uso específico do super-usuário e são os seguintes: **wuser**, **wim**, **wmm**, **wdm**. O segundo grupo tem à disposição do usuário habilitado os seguintes comandos: **wld** e **wlw**

O comando **wuser** permite ao super-usuário cadastrar um usuário do saa-WORM e tem a seguinte sintaxe: **wuser** <identificação do usuário> [área]. A área (em megabytes) é um parâmetro opcional e possibilita ao super-usuário definir o uso do disco worm de acordo com a necessidade do usuário. Isto permite a adoção de uma política de distribuição de área de acordo com a classe do usuário. O pseudo código para o **wuser** é o seguinte:

```
faça
  se usuário é o super-usuário, então:
    lê <identificação do usuário id_user> [área];
    inclui dados do usuário no cadastro de usuários, arquivo wusers;
  fimse
fim
```

O comando **wim** é utilizado pelo super-usuário para iniciar uma mídia (disquete) para uso no saa-WORM. O processo de inicialização grava em toda a mídia (disquete) um padrão de byte (255) com o propósito de simular uma mídia de WORM. Com esta gravação é possível saber se um bloco já foi gravado ou não, sendo possível identificar as três regiões da mídia. A sintaxe para o comando **wim** é a seguinte: **wim** <dispositivo MINIX> <tamanho em blocos> <identificação para a mídia>

O dispositivo MINIX pode ser qualquer um dos dispositivos do MINIX utilizados para sistemas de arquivos em disquetes: /dev/fd0, /dev/fd1, /dev/at0 ou /dev/at1. O tamanho em blocos reflete o tamanho do disquete a ser utilizado: 360Kb ou 1,2Mb. A identificação para a mídia é de livre escolha do super-usuário. O pseudo código é o seguinte:

```
faça
  se usuário é o super-usuário então:
    lê <dispositivo> <tamanho> <identificação>
    se dispositivo está disponível, então:
      para n=1 até <tamanho>, faça:
        grava bloco[n] com padrão de byte 255
      fimp
    fimse
  fimse
fim
```

O comando **wmm** é utilizado pelo super-usuário toda vez que houver necessidade de montar uma nova mídia (disquete) no saa-WORM. A sintaxe é a seguinte: **wmm** <dispositivo MINIX>

Sendo que o dispositivo MINIX é um dos dispositivos relacionados acima no comando **wim** e onde está a mídia (disquete) a ser montada para uso. O pseudo código é o seguinte:

```
faça
  se usuário é o super-usuário então:
    lê <dispositivo>
    se dispositivo está disponível, então:
      inclui dispositivo no arquivo de mídias em uso, worm.mid
    fimse
  fimse
fim
```

Para desmontar (retirar de uso) uma mídia (disquete) do **saa_WORM**, o super-usuário necessita do comando **wdm** que tem a seguinte sintaxe:
wdm <dispositivo MINIX>

Sendo que o dispositivo MINIX é um dos dispositivos relacionados acima no comando **wim** e onde está a mídia (disquete) a ser desmontada. O pseudo código é o seguinte:

```
faça
  se usuário é o super-usuário então:
    lê <dispositivo>
    se dispositivo está em uso pelo saa-WORM, então:
      desativa este dispositivo no arquivo de mídias em uso, worm.mid
    fimse
  fimse
fim
```

O comando **wld**, que pertence ao segundo grupo de comandos de apoio, permite ao usuário do **saa-WORM** listar o diretório do **WORM** contendo todas as entradas de todos os arquivos gravados nos discos **WORM** (disquetes). A sintaxe para este comando é a seguinte: **wld**

O pseudo código para o **wld** é o seguinte:

```
faça
  abre o arquivo diretório do WORM;
  enquanto existir entrada de arquivo faça;
    listar a entrada do arquivo;
  fimenq
fim
```


O comando **wlw** tem função idêntica ao **wld**, porém lista somente as entradas no diretório correspondentes aos arquivos que estão na mídia (disquete) ativa. A sintaxe é a seguinte: **wlw**

O pseudo código para o **wlw** é o seguinte:

```
faça
    enquanto existir entrada de arquivo no diretório da mídia (disquete) em uso,
        faça;
        listar a entrada do arquivo;
    fimenq
fim
```

No Apêndice C estão resumidos todos os comandos do saa-WORM.

6.7 TÓPICOS DE DESTAQUE DA IMPLEMENTAÇÃO

A proposta de utilização de sub-diretórios para implementação das listas de arquivos a serem transferidos para o disco WORM, resolve dois problemas básicos do saa-WORM: o primeiro com respeito à questão de segurança dos arquivos que passaram a ter uma duplicata no próprio disco magnético e a segunda relacionada com a necessidade de agilizar a transferência de arquivos do disco magnético para o disco WORM. Como a cópia do arquivo no próprio disco magnético é bastante rápida, passou a servir também como um "cache" para o saa-WORM, deixando livre o arquivo original logo após a cópia.

Com a utilização dos sub-diretórios crítico, cópia e desuso correspondendo as listas de arquivos crítico, normais e normais não acessados há mais de 90 dias, é muito fácil identificar por exemplo quais são os arquivos críticos em um determinado instante. Basta consultar as entradas do sub-diretório crítico. O mesmo procedimento pode ser adotado para se saber quais são os arquivos não acessados há mais de 90 dias e arquivos copiados por solicitação de usuário habilitado.

A duplicação do arquivo não acessado a mais de 90 dias no próprio disco magnético não tem sentido, uma vez que o objetivo do saa-WORM é de eliminar estes arquivos do disco magnético liberando espaço e não comprometendo ainda mais. Tendo em vista este problema foi proposto então a duplicação somente da entrada do arquivo no sub-diretório desuso correspondente a lista de arquivos não acessados há mais de 90 dias. Uma maneira eficiente encontrada para isto foi criar uma ligação ("link") no sub-diretório desuso ao arquivo original. Assim também é

válido o uso do diretório desuso para indicação dos arquivos removidos para o disco WORM.

O uso de sub-diretórios correspondentes as listas apesar de facilitar o gerenciamento do saa-WORM, trouxe um problema com respeito a unicidade da entrada do arquivo no sub-diretório. Considerando-se que é perfeitamente possível, em um sistema de arquivos, ter arquivos com o mesmo nome mas em diretórios distintos, uma solução segura deveria ser proposta. Uma solução alternativa seria a criação da mesma estrutura do arquivo ("path") dentro do sub-diretório. Porém esta alternativa é muito dispendiosa. A solução proposta é muito mais simples e segura, implica no uso do próprio i-node do arquivo como entrada (identificação) do arquivo no sub-diretório. O uso do i-node garante a unicidade do arquivo na lista (sub-diretório) uma vez que ele é único em um sistema de arquivos. Foi acrescido ao número do i-node uma letra "A" como prefixo, meramente por estética.

Como todo dispositivo de memória, o disco WORM também não é ilimitado. Por isso, julgou-se conveniente a adoção de uma política de distribuição de área do disco WORM para os usuários habilitados. Deste modo é possível ao administrador do sistema (super-usuário) atribuir cotas de área do disco WORM de acordo com a necessidade e importância do usuário para o sistema. Esta restrição evidentemente só se aplica aos arquivos críticos e arquivos normais copiados para o disco WORM através de solicitação do usuário e não aos arquivos normais copiados por longevidade. A Figura 6.1 mostra a representação gráfica do saa-WORM

Fig.6.1 Estrutura e organização do saa-WORM

6.8 TESTES

Para a realização dos testes do saa-WORM é necessário apenas de um micro-computador compatível com o IBM PC/XT com duas unidades de disquetes, e os disquetes de BOOT, ROOT eUSR do sistema MINIX. Este último foi previamente preparado com o saa-WORM. Visando explorar todos os mecanismos e procedimentos propostos para o sistema preparou-se um roteiro para os testes que é apresentado a seguir:

Instruções Para Uso do Sistema Auxiliar de Arquivos WORM em Disquete

01. Instalar disquete de BOOT no drive a: e iniciar re-boot atendendo as solicitações do MINIX
02. Quando o MINIX solicitar login, abrir o sistema como "su", conta **root**
03. Ativar o procedimento shell para iniciar o saa-WORM: **sh /usr/worm/winstala**
04. Cadastrar um usuário do sistema MINIX, ex.: lafbezer:
mined /etc/passwd
lafbezer::9:3::/usr/lafbezer:
^Z
05. Cadastrar usuário do saa-WORM (habilitar usuário) super-usuário (root) e lafbezer:
/usr/worm/wuser root
/usr/worm/wuser/lafbezer
06. Preparar mídia (disquete) como disco WORM:
/usr/worm/wim /dev/at1 1200 disc001
07. Alterar data para um ano atrás a fim de demonstrar o processo dos arquivos de longevidade
Consultar data atual: **date**
Alterar data: **date MMDDAAHHMMSS**
08. Encerrar sessão do super-usuário: **exit**
09. Abrir sessão (login) como lafbezer: **login lafbezer**
10. Remover arquivos do subdiretório tmp: **rm tmp/***
11. Copiar arquivos t* do diretório corrente para o sub-diretório tmp: **cp t* tmp**
12. Terminar sessão: **exit**
13. Abrir sessão para o super-usuário: **login root**
14. Corrigir a data do sistema para a data atual: **date**
date MMDDAAHHMMSS
15. Confirmar que a lista de arquivos de longevidade está vazia (sub-diretório desuso):
ls -asl /usr/worm/desuso
16. Verificar área em uso e livre do sistema MINIX: **du; df**
17. Localizar todos os arquivos não usados a mais de 90 dias no /usr e indicar na lista desuso
/usr/worm/wdes
18. Verificar quais arquivos foram encontrados: **ls -asl /usr/worm/desuso**
19. Verificar área em uso e livre do sistema MINIX: **du; df**
20. Encerrar sessão: **exit**
21. Abrir sessão do usuário lafbezer: **login lafbezer**
22. Confirmar lista vazia de copia: **ls -asl /usr/worm/copia**
23. Copiar um arquivo do usuário para o WORM: **/usr/worm/wcp djlist.c**
24. Verificar o estado atual da lista copia: **ls -asl /usr/worm/copia**
25. Verificar área em uso e livre do sistema MINIX: **du; df**
26. Definir um arquivo do usuário como crítico: **/usr/worm/wci itoafmt.c**
27. Verificar o estado do diretório do WORM:
(no disco magnético): **/usr/worm/wld**

- (no disquete-WORM): /usr/worm/wlw
28. Encerrar sessão
 29. Abrir sessão para o super-usuário: login root
 30. Ativar o gerente de transferência do saa-WORM: /usr/worm/wgt
 31. Cancelar processo wgt após ter transferido todas as listas: kill <pid-wgt>
F1 - informa a lista de processos
 32. Verificar o estado do diretório do WORM:
(no disco magnético): /usr/worm/wld
(no disquete-WORM): /usr/worm/wlw
 33. Verificar o estado das listas: crítico, cópia e desuso:
ls -asl /usr/worm/critico
ls -asl /usr/worm/copia
ls -asl /usr/worm/desuso
 34. Verificar área em uso e livre: du; df
 35. Verificar o estado do sub-diretório /usr/lafbezer/tmp: ls -asl /usr/lafbezer/tmp
 36. Encerrar sessão
 37. Abrir sessão como lafbezer: login lafbezer
 38. Copiar o arquivo tmp/twgr para o diretório lixo (Re-instalação automática de cópia):
cp tmp/twgr lixo
comparar com o original em /usr/lafbezer: cmp tmp/twgr /usr/lafbezer/twgr
idem para tmp/twle
 39. Re-instalar cópia de um arquivo qualquer transferido para o WORM por solicitação do usuário:
cp itoafmt.c lixo (para comparação)
rm itoafmt.c
/usr/worm/wrc itoafmt.c
cmp itoafmt.c lixo/itoafmt.c
 40. Encerrar a sessão
 41. Fim.

A execução deste roteiro exemplifica a utilização e permite a verificação de todos os comandos discutidos neste capítulo.

6.9 COMENTÁRIOS

A implementação do saa-WORM permitiu demonstrar sua viabilidade, apesar de não ter sido possível a utilização de um disco ótico WORM. Mesmo assim todos os mecanismos definidos para o saa- WORM, foram explorados e implementados utilizando-se uma unidade de disquete, simulando um disco ótico. A escolha do disquete se deve ao fato de que é um recurso disponível à maioria dos sistemas

operacionais, é removível como o disco ótico e é fácil de simular única escrita como o WORM. Portanto, apesar da capacidade de armazenagem ser muito menor do que a mídia WORM, o disquete oferece condições satisfatória para implementar o sistema proposto.

A implementação do saa-WORM foi realizada em linguagem de programação C, utilizando-se o sistema operacional MINIX. Para que o saa-WORM atendesse aos propósitos deste trabalho com respeito ao aumento de segurança e disponibilidade de espaço, servindo como um sistema auxiliar de arquivos ao sistema de arquivos do MINIX, foi necessário desenvolver vários processos de suporte e gerenciamento de arquivos a serem transferidos do disco magnético para o disco WORM (disquete) de acordo com o tipo atribuído aos arquivos (crítico, normal solicitado por usuário habilitado e normal não acessado há mais de 90 dias). Para que o sistema saa-WORM fosse o mais transparente possível ao usuário, houve necessidade também de alterações dos procedimentos OPEN e CLOSE do sistema de arquivos do MINIX

A abordagem básica da implementação constitui-se de um processo gerenciador de transferência entre o disco magnético e o disco WORM (disquete). Este processo permanece todo o tempo em "background" gerenciando as listas de arquivos a serem copiados para o WORM.

As listas de gerenciamento de arquivos a serem transferidos para o WORM (disquete) (crítico, normal solicitado por usuário habilitado e normal não acessado há mais de 90 dias) foram implementadas como sub-diretórios (crítico, cópia e desuso). Esta abordagem proporcionou ao saa-WORM a solução de dois problemas básicos enfocados neste trabalho: o primeiro que diz respeito a questão de segurança dos arquivos, que passaram a ter uma duplicata no próprio disco magnético e o segundo relacionado com a necessidade de agilizar a transferência de arquivos do disco magnético para o disco WORM (disquete). Pode-se afirmar que a transferência é agilizada porque a cópia é feita pelo próprio sistema de arquivos (do MINIX) através do comando `cp` no próprio disco magnético (em um dos sub-diretórios correspondente a lista de arquivos). Isto libera o arquivo original rapidamente para uso pelo usuário e deixa a cópia a disposição do gerenciador de transferência para escalar o processo copiador do saa-WORM quando for conveniente. Assim além da duplicata do arquivo agir como medida de segurança pode-se considerar também como uma forma de "cache" para o saa-WORM.

Outro aspecto de destaque do uso dos sub-diretórios é a facilidade de identificar por exemplo quais são os arquivos críticos do sistema de arquivos, bastando para isso consultar quais são as entradas do diretório do sub-diretório crítico.

Considerando-se a relevância do comprometimento de espaço do sistema de arquivos por parte dos arquivos não acessados há mais de 90 dias, propôs-se uma abordagem mais eficiente para este caso através da criação de uma ligação "link" no sub-diretório desuso ao arquivo original. Deste modo apenas a identificação do arquivo é duplicada e não o seu conteúdo.

O uso de sub-diretórios correspondentes às listas apesar de facilitar o gerenciamento do saa-WORM, trouxe um problema com respeito a unicidade da entrada do arquivo no sub-diretório. Considerando-se que é perfeitamente possível, em um sistema de arquivos, ter arquivos com o mesmo nome mas em diretórios distintos, uma solução segura deveria levar isto em consideração. Uma solução possível seria a criação da mesma estrutura do arquivo ("path") dentro do sub-diretório. Porém esta alternativa é muito dispendiosa. A solução proposta é muito mais simples e segura, implica no uso do próprio i-node do arquivo como entrada (identificação) do arquivo no sub-diretório. O uso do i-node garante a unicidade do arquivo na lista (sub- diretório) uma vez que ele é único em um sistema de arquivos. Foi acrescentado ao número do i-node uma letra "A" como prefixo, meramente por estética.

Quanto aos procedimentos implementados, é possível agrupá-los de acordo com a interferência do usuário, podendo ser: procedimentos transparentes ao usuário, comandos disponíveis ao usuário e outros comandos de apoio ao usuário.

Os procedimentos transparentes ao usuário são os que independem de qualquer participação do usuário e são responsáveis pelo gerenciamento de transferência de arquivos entre o disco magnético e disco WORM (disquete), remoção dos arquivos não acessados há mais de 90 dias, cópia de arquivos críticos e re-instalação (transferência de arquivos antigos do disco WORM (disquete) para o disco magnético).

O segundo grupo de comandos permite ao usuário habilitado indicar seus arquivos críticos, copiar arquivos quaisquer e re-instalar uma cópia de um arquivo do disco WORM (disquete) para o disco magnético.

Os comandos de apoio são destinados ao administrador do sistema (super-usuário) e ao usuário habilitado. Permitem cadastrar um usuário para uso do WORM, inicializar uma mídia (disquete), montar uma mídia para uso, desmontar uma mídia e listar diretórios do saa-WORM.

Como todo dispositivo de memória, o disco WORM também não é ilimitado. Por isso, julgou-se conveniente a adoção de uma política de distribuição de área do disco WORM para os usuários habilitados. Deste modo é possível ao administrador do sistema (super-usuário) atribuir cotas de área do disco WORM de acordo com a necessidade e classe do usuário. Esta restrição evidentemente só se aplica aos arquivos críticos e arquivos normais copiados para o disco WORM através de solicitação do usuário e não aos arquivos normais copiados por longevidade.

COMENTÁRIOS FINAIS

7.1 REVISÃO BIBLIOGRÁFICA

A revisão bibliográfica apresentada demonstrou a importância dos sistemas distribuídos [SV84] [TA85] [CO88] [MU89] não só pelo aspecto econômico mas também pelo desempenho e recursos oferecidos aos usuários. Estes sistemas oferecem ao usuário um grande potencial de computação porque colocam a sua disposição recursos transparentes e compartilhados por vários usuários. Estes recursos podem ser diferentes tipos de servidores (impressão, arquivos, discos óticos, etc) que são acessados através de uma rede de computadores. Um aspecto importante dos sistemas distribuídos em relação aos sistemas centralizados é a tolerância a falhas. No sistema distribuído, a falha de uma parte do sistema não implica na impossibilidade de utilização de todo o sistema, uma vez que uma outra parte do sistema pode assumir as funções da parte com defeito.

Considerando-se os trabalhos de [ME84] [KI86] [TA89] [WI91] ficou clara a importância das redes de comunicação. A topologia da rede e o protocolo de acesso ao meio são características essenciais para o bom desempenho do sistema distribuído. O meio de comunicação pode variar desde um simples par trançado até a utilização de fibra ótica. A escolha do meio de comunicação, da topologia e do protocolo de acesso ao meio mais adequado depende dos objetivos propostos para a rede e de fatores tais como: custo, desempenho e alcance.

Um servidor essencial para sistemas distribuídos é o servidor de arquivos que oferece a possibilidade de um armazenamento centralizado, facilitando a troca de informações, a mobilidade dos usuários e a adoção de estações de trabalho sem disco. Vários trabalhos como os propostos por [SV84] [TA85] [LE90] [SA90a] [SA90] discutem aspectos de projeto de diversos servidores de arquivos. Ressaltam

características importantes de interface de cliente que dependem da aplicação e do ambiente do sistema.

Uma vez que o propósito do servidor de arquivos é oferecer um tipo particular de serviço em atendimento a um pedido de um cliente, é necessário que se tenha primitivas que realizem operações, tais como: create-file, read-data, write-data e delete-file [SV84] [TA85].

Questões de segurança e concorrência nos servidores de arquivos foram tratadas com mecanismos de transações atômicas. E para melhorar a confiabilidade dos servidores de arquivos foram propostos procedimentos simples como fazer cópias periódicas até os mais elaborados que envolvem arquivos robustos, por meio de técnicas de replicação dos arquivos [TA85] [HU91].

Com o advento dos discos óticos de única escrita, o armazenamento estruturado de registro "log" tornou-se atrativo para projeto de sistemas de arquivos confiáveis com suporte de versões como o sistema SWALLOW [SV81] o sistema de arquivos de única escrita proposto por GARFINKEL [GA91] e o sistema de arquivo de acesso aleatório para discos óticos de única escrita proposto por GAIT [GA88]. Estes trabalhos permitiram abordar aspectos relevantes da estrutura física dos discos óticos tais como, materiais que os constituem, forma de gravação e leitura de dados e resistência ao meio ambiente.

A gravação de dados nos discos óticos de única escrita é organizada em unidades chamadas blocos, que são unidades indivisíveis e contínuas de armazenagem. Quando um bloco é gravado no disco, o código de correção de erro correspondente ao bloco também é gravado em uma posição convencional. Por esta razão, blocos individuais do disco, uma vez gravados, nunca são regravados, mesmo que ainda tenha espaço no bloco [GA88].

Foram tratadas também algumas técnicas propostas para a armazenagem em discos óticos, que podem ser: somente anexar no final; somente copiar; armazenar os dados no disco ótico e informações auxiliares em discos magnéticos e a última empregar técnicas de árvores B para aplicações de base de dados [GA88].

Os exemplos apresentados de sistemas de arquivos utilizando os discos óticos de única escrita fortalecem a viabilidade e mostram a importância do trabalho desenvolvido.

A proposta de utilização de discos óticos, como servidores para arquivamento, apresentada por Santana em [SA90], foi adotada como base para o desenvolvimento do saa-WORM.

7.2 SISTEMA AUXILIAR DE ARQUIVOS WORM

Este trabalho foi proposto em consonância com os objetivos do grupo de sistemas distribuídos do ICMSC-USP em investigar novas propostas para servidores de arquivos, em termos de desempenho, segurança e disponibilidade.

Um dos problemas de segurança de informações existentes em qualquer sistema computacional é a confiabilidade. Este problema teve algumas soluções propostas para os sistemas distribuídos, dentre elas, uma muito simples que consiste em fazer cópias periódicas dos sistemas de arquivos ("backup's"); uma outra solução envolve arquivos robustos através de replicação de arquivos por meio de arquivos imagem ("mirror disk") e uma terceira solução que é um aperfeiçoamento da segunda permite que o servidor de arquivos ofereça armazenagem estável e transações atômicas [TA85]. A primeira abordagem deixa totalmente a responsabilidade de copiar e restaurar o sistema de arquivos para o usuário. Se o período entre as cópias for muito longo e se o servidor de arquivos falhar, as informações armazenadas após a última cópia serão perdidas. A segunda e terceira já são mais elaboradas, mais caras e tornam a estrutura do servidor de arquivos mais complexa.

O propósito do sistema auxiliar de arquivos, baseado em disco WORM, é de realizar para os servidores de arquivos os procedimentos de transferência de arquivos [SA81] dos discos magnéticos para o WORM automaticamente. Esta transferência se faz necessário por questões de segurança e de disponibilidade de espaço no servidor de arquivos. Com relação a segurança o sistema auxiliar de arquivos baseado em disco WORM cuida para que determinados arquivos sejam copiados para o disco WORM, automaticamente e o mais breve possível, toda vez que forem alterados. Com o propósito de aumentar a disponibilidade de espaço do servidor de arquivos, o sistema auxiliar de arquivos baseado em disco WORM, dispõe de procedimentos que monitoram todos arquivos do sistema de arquivos, com atenção especial aos arquivos com baixa probabilidade de acesso [SA87] [SA81]. É possível também a participação de usuários privilegiados que podem solicitar ao sistema auxiliar de arquivos a cópia de um arquivo qualquer para o disco WORM. A recuperação (ou re-instalação) dos arquivos do disco WORM para o disco magnético

pode ser feita também de maneira transparente para o usuário ou através de solicitação do próprio usuário.

Mostrou-se através de levantamento de alguns sistemas de arquivos, o comprometimento de espaço em disco por arquivos não acessados a mais de 180 dias. Apesar do avanço tecnológico proporcionar discos magnéticos cada vez mais poderosos em capacidade e tempo de acesso, cresce também o número de usuários, conseqüentemente, aumenta a demanda por espaço. Analisando-se dados coletados em diferentes sistemas, observou-se que na média a área utilizada corresponde a 80% da área total enquanto que a área comprometida com arquivos não acessados a mais de 180 dias corresponde a 68% (em média), da área em uso. Os sistemas de menor capacidade de armazenamento são os mais comprometidos com os arquivos que não são acessados a mais de 180 dias. Considerando-se a importância da liberação desta área e a média de área em uso por usuário, pode-se estimar o acréscimo de usuários para cada sistema de arquivos. O número de usuários que é possível aumentar corresponde exatamente ao percentual da área comprometida pela área em uso.

Considerando-se os aspectos de segurança e disponibilidade de espaço do servidor de arquivos, três tipos de arquivos para gravação em WORM foram propostos: **crítico, normal e morto**.

Esta classificação evidencia a importância da cópia em WORM como medida de segurança - replicação do arquivo em dispositivo mais robusto. Quanto a disponibilidade de espaço do servidor de arquivos, foi proposto um mecanismo de remoção baseado no tempo de permanência dos arquivos nos discos magnéticos sem acesso. Considerou-se como fator determinante para a remoção dos arquivos normais para o WORM o tempo de 90 dias. Esta remoção pode significar um ganho de espaço importante para o servidor de arquivos, como pode ser visto pelos sistemas de computação analisados na seção 6.2, onde estes arquivos comprometem entre 43% a 83% da área em uso.

É importante ressaltar que a remoção dos arquivos normais por longevidade não impõe nenhum obstáculo a futuros acessos a estes arquivos. Contudo, pode haver um aumento do tempo de resposta em decorrência da necessidade de re-instalar o arquivo no disco magnético. O saa-WORM mantém a entrada original do arquivo no disco magnético e cria um "link" ao arquivo para permitir a re-instalação do arquivo quando for necessário. Este processo de remoção do arquivo normal (longevidade) para o WORM e re-instalação no disco magnético é totalmente transparente para as duas categorias de usuários, habilitados e não habilitados. A transferência dos

arquivos críticos, uma vez definidos pelos usuários habilitados, é também transparente a todos os usuários e ocorre sempre após o uso com alteração do arquivo crítico.

A estrutura do sistema de arquivos proposta para o disco WORM considerou o disco dividido em três regiões: de arquivos, livre e de diretório. Uma vez definida a estrutura e os mecanismos do saa-WORM foi possível propor a implementação visando demonstrar a viabilidade do sistema auxiliar de arquivos WORM, apesar de não se ter conseguido o dispositivo de disco ótico WORM pretendido. Mas, mesmo assim todos os mecanismos definidos para o saa-WORM, foram explorados e implementados utilizando-se uma unidade de disquete, simulando um disco ótico. A escolha do disquete se deve ao fato de que é um recurso disponível a maioria dos sistemas operacionais, é removível como o disco ótico e é fácil de simular única escrita como o WORM.

Para que o saa-WORM atendesse aos propósitos deste trabalho com respeito ao aumento de segurança e disponibilidade de espaço, servindo como um sistema auxiliar de arquivos ao sistema de arquivos do MINIX, foi necessário desenvolver vários processos de suporte e gerenciamento de arquivos a serem transferidos do disco magnético para o disco WORM (disquete) de acordo com o tipo atribuído aos arquivos (crítico, normal solicitado por usuário habilitado e normal não acessado há mais de 90 dias). Para que o sistema saa-WORM fosse o mais transparente possível ao usuário, houve necessidade também de alterações dos procedimentos OPEN e CLOSE do sistema de arquivos do MINIX

A abordagem básica da implementação constitui-se de um processo gerenciador de transferência entre o disco magnético e o disco WORM (disquete). Este processo permanece todo o tempo em "background" gerenciando as listas de arquivos a serem copiados para o WORM.

As listas de gerenciamento de arquivos a serem transferidos para o WORM (disquete) (crítico, normal solicitado por usuário habilitado e normal não acessado há mais de 90 dias) foram implementadas como sub-diretórios (crítico, cópia e desuso). Esta abordagem proporcionou ao saa-WORM a solução de dois problemas básicos enfocados neste trabalho: o primeiro que diz respeito a questão de segurança dos arquivos, que passaram a ter uma duplicata no próprio disco magnético e o segundo relacionado com a necessidade de agilizar a transferência de arquivos do disco magnético para o disco WORM (disquete). Pode-se afirmar que a transferência é agilizada porque a cópia é feita pelo próprio sistema de arquivos (do MINIX) através do comando `cp` no próprio disco magnético (em um dos sub-

diretórios correspondente a lista de arquivos). Isto libera o arquivo original rapidamente para uso pelo usuário e deixa a cópia a disposição do gerenciador de transferência para escalar o processo copiador do saa-WORM quando for conveniente. Assim além da duplicata do arquivo agir como medida de segurança pode-se considerar também como uma forma de "cache" para o saa-WORM.

Outro aspecto de destaque do uso dos sub-diretórios é a facilidade de identificar por exemplo quais são os arquivos críticos do sistema de arquivos, bastando para isso consultar quais são as entradas do diretório do sub-diretório crítico.

O uso de sub-diretórios correspondentes as listas apesar de facilitar o gerenciamento do saa-WORM, trouxe um problema com respeito a unicidade da entrada do arquivo no sub-diretório. A solução proposta é simples e segura e utiliza o próprio i-node do arquivo como entrada (identificação) do arquivo no sub-diretório. O uso do i-node garante a unicidade do arquivo na lista (sub- diretório) uma vez que ele é único em um sistema de arquivos. Foi acrescido ao número do i-node uma letra "A" como prefixo, meramente por estética.

Quanto aos procedimentos implementados, é possível agrupá-los de acordo com a interferência do usuário, podendo ser: procedimentos transparentes ao usuário, comandos disponíveis ao usuário e outros comandos de apoio ao usuário.

Os procedimentos transparentes ao usuário são os que independem de qualquer participação do usuário e são responsáveis pelo gerenciamento de transferência de arquivos entre o disco magnético e disco WORM (disquete), remoção dos arquivos não acessados há mais de 90 dias, cópia de arquivos críticos e re-instalação, do disco WORM (disquete) para o disco magnético, de arquivos não acessados há mais de 90 dias.

Os comandos disponíveis aos usuários permitem ao usuário habilitado indicar seus arquivos críticos, copiar arquivos quaisquer e re-instalar uma cópia de um arquivo do disco WORM (disquete) para o disco magnético.

Os comandos de apoio são destinados ao administrador do sistema (super-usuário) e ao usuário habilitado. Permitem cadastrar um usuário para uso do WORM, inicializar uma mídia (disquete), montar uma mídia para uso, desmontar uma mídia e listar diretórios do saa-WORM.

Como todo dispositivo de memória, o disco WORM também não é ilimitado. Por isso, julgou-se conveniente a adoção de uma política de distribuição de área do disco WORM para os usuários habilitados. Deste modo é possível ao administrador do sistema (super-usuário) atribuir cotas de área do disco WORM de acordo com a necessidade e importância do usuário para o sistema. Esta restrição evidentemente só se aplica aos arquivos críticos e arquivos normais copiados para o disco WORM através de solicitação do usuário e não aos arquivos normais copiados por longevidade.

7.3 CONTRIBUIÇÃO DO TRABALHO PROPOSTO

Considerando-se a importância dos tópicos enfocados neste trabalho com relação à segurança, disponibilidade das informações de sistemas distribuídos baseados em rede local de computadores, acredita-se que a utilização de disco ótico WORM como solução preventiva para estes problemas seja satisfatória tendo-se em vista as características destes discos. Em relação a outros dispositivos, tais como: discos e fitas magnéticas, a utilização dos discos óticos apresentam diversas vantagens:

- não se alteram em presença de campos magnéticos,
- grande capacidade de armazenamento,
- baixo custo por byte armazenado,
- gravação única, i.e., não permitem que seu conteúdo seja alterado após ter sido gravado,

Acredita-se que a contribuição principal deste trabalho está na abordagem simples e segura proposta com o sistema auxiliar de arquivos baseado em disco WORM. O propósito do sistema auxiliar de arquivos, é de realizar para os servidores de arquivos os procedimentos de transferência de arquivos [SA81] dos discos magnéticos para o WORM automaticamente. Esta transferência se faz necessário por questões de segurança e de disponibilidade de espaço ao servidor de arquivos. Com relação a segurança o sistema auxiliar de arquivos baseado em disco WORM cuida para que determinados arquivos sejam copiados para o disco WORM, automaticamente e o mais breve possível, toda vez que forem alterados. Com o propósito de aumentar a disponibilidade de espaço do servidor de arquivos, o sistema auxiliar de arquivos baseado em disco WORM, dispõe de procedimentos que monitoram todos arquivos do sistema de arquivos, agindo em função da data do último acesso do arquivo [SA87] [SA81]. É possível também a participação de usuários privilegiados

que podem solicitar ao sistema auxiliar de arquivos a cópia de um arquivo qualquer para o disco WORM. A recuperação (ou re-instalação) dos arquivos do disco WORM para o disco magnético pode ser feita também de maneira transparente para o usuário ou através de solicitação do próprio usuário.

7.4 CONTINUIDADE DO TRABALHO

Considerando-se a limitação imposta principalmente na implementação pelos recursos disponíveis, espera-se que seja possível implementar este sistema auxiliar de arquivos baseados em disco ótico WORM em ambientes computacionais mais elaborados, onde se tenha oportunidade de explorar o próprio disco WORM e os servidores de arquivos completos.

A importância de sistemas de gerenciamento de armazenagem para os sistemas computacionais distribuídos é evidenciada pelo número de fabricantes de software para "backup's" de redes locais. Esses fabricantes estão adicionando aos seus produtos, recursos distribuídos e hierarquicos de gerenciamento de armazenagem, utilizando-se vários dispositivos, replicação de discos, fitas DAT e discos óticos [MO93].

A utilização de disco ótico atualmente é pequena por tratar-se de uma tecnologia emergente. Contudo, com a evolução cada vez mais expressiva da eletrônica, os sistemas de computação estão se tornando cada vez mais sofisticados, permitindo a exploração de recursos de vídeo, áudio e formação de grandes redes de computadores. Estes recursos de vídeo e áudio que permitem a animação gráfica por computador tem feito do disco ótico um recurso imprescindível.

que podem solicitar ao sistema auxiliar de arquivos a cópia de um arquivo qualquer para o disco WORM. A recuperação (ou re-instalação) dos arquivos do disco WORM para o disco magnético pode ser feita também de maneira transparente para o usuário ou através de solicitação do próprio usuário.

7.4 CONTINUIDADE DO TRABALHO

Considerando-se a limitação imposta principalmente na implementação pelos recursos disponíveis, espera-se que seja possível implementar este sistema auxiliar de arquivos baseados em disco ótico WORM em ambientes computacionais mais elaborados, onde se tenha oportunidade de explorar o próprio disco WORM e os servidores de arquivos completos.

A importância de sistemas de gerenciamento de armazenagem para os sistemas computacionais distribuídos é evidenciada pelo número de fabricantes de software para "backup's" de redes locais. Esses fabricantes estão adicionando aos seus produtos, recursos distribuídos e hierarquicos de gerenciamento de armazenagem, utilizando-se vários dispositivos, replicação de discos, fitas DAT e discos óticos [MO93].

A utilização de disco ótico atualmente é pequena por tratar-se de uma tecnologia emergente. Contudo, com a evolução cada vez mais expressiva da eletrônica, os sistemas de computação estão se tornando cada vez mais sofisticados, permitindo a exploração de recursos de vídeo, audio e formação de grandes redes de computadores. Estes recursos de vídeo e audio que permitem a animação gráfica por computador tem feito do disco ótico um recurso imprescindível.

BIBLIOGRAFIA

- [BI80] BIRRELL, A.D. and NEEDHAM, R.M.. A Universal File Server. *IEEE Transactions on Software Engineering*, v.SE-6, n.5, p.450-453, Sep., 1980.
- [CE91] CELENTANO, M. A.. CD-ROM O Futuro no Presente. São Paulo, Editora Érica, 1991.
- [CH86] CHRISTODOULAKIS, S. and FALOUTSOS, C.. *Design and Performance Considerations for an Optical Disk-Based, Multimedia Object Server*. IEEE Computer, p.45-56, Dec. 1986.
- [CO88] COULOURIS, G. F. and DOLLIMORE, J.. *Distributed Systems: Concepts and Design*. Wokingham, Addison-Wesley, 1988.
- [FU84] FUJITANI, L.. Laser Optical Disk: The Coming Revolution in On-Line Storage. *Comm. ACM*, v. 27, n. 6, p. 546-554, Jun. 1984
- [FU92] FURQUIM, R. T. M.. *Servidor de Impressão para um Ambiente Computacional Distribuído*. Dissertação de Mestrado, ICMSC-USP, São Carlos, Nov/1992.
- [GA88] GAIT, J.. The Optical File Cabinet: A Random-Access File System for Write-Once Optical Disks. *IEEE Computer*, v. 21, n. 6, p. 11-22, Jun. 1988.
- [GA91] GARFINKEL, S.. Designing a Write-Once File System. *Dr. Dobbs's Journal*, p. 78-86, Jan. 1991.
- [HU91] HURWICZ, M.. Selecting Server Storage. *LAN Technology*, p.45-54, Aug. 91.
- [KI86] KIRNER, C.. Desenvolvimento de Suporte Básico para Sistemas Operacionais Distribuídos. Tese de Doutorado, COPPE/UFRJ, Rio de Janeiro, Ago. 1986.
- [LE90] LEVY, E. and SILBERSCHATZ, A.. Distributed File Systems: Concepts and Examples. *ACM Computing Surveys*, v. 22, n. 4, p. 321-374, Dec. 1990.
- [ME84] MENASCÉ, D. A. e SCHWABE, D.. *Redes de Computadores, Aspectos Técnicos e Operacionais*. Rio de Janeiro, Editora Campus, 1984.
- [MI88] MIKRO. *Compact Disk com Superfície Orgânica*. Eletrônica, Informática, Automação e Telecomunicações. n.8, p.40-51, Set. 88.
- [MO93] Your Next LAN Storage Could Be A Mainframe. *DATAMATION*, Mar/93, p.71-74.
- [MU89] MULLENDER, S. J.. *Distributed Systems*. New York, Addison-Wesley, 1989.
- [NE84] NELSON, D.L. and LEACH, P.. The Architecture and Applications of the Apollo Domain. *IEEE Computer Graphics and Applications*, p.58-66, Abr. 1984.
- [OL91] OLIVEIRA, E. T.. O Laser Assume a Vanguarda. *IPESI Negócios & Informática*, p. 24-29, Out/Nov 1991.
- [QU90] QUILAN, S.. A Cached WORM File System. *Software Practice and Experience*. v. 21(12), p. 1289-1299, Dec. 1991.
- [RO91] ROSSATO, A.P.G.S.. *Servidor de Arquivos Para Sistemas Distribuídos*. Minidissertação de Mestrado, ICMSC-USP, São Carlos, 1991.

- [SA87] SANTANA, M.J.. Analysis of The TRICE Filestore Statistical Data. University of Southampton, Dept. of Electronics and Computer Science, 1987.
- [SA90] SANTANA, M.J.. *An Advanced Filestore Architecture for multiple-LAN distributed computing system*. PhD Thesis, Department of Eletronics and Computer Science, University of Southampton, UK, 1990.
- [SA90a] SANTANA, R.H.C.. *Performance evaluation of LAN-based file servers*. PhD Thesis, Department of Eletronics and Computer Science, University of Southampton, UK, 1990.
- [SA81] SATYANARAYANAN, M.. A Study of File Sizes and Funcional Lifetimes. *Proceedings Eighth Symposium on Operating Systems Principles*. p. 96-108, Dec. 1981
- [SV81] SVOBODOVA, L.. A Reliable Object-Oriented Repository for a Distributed Computer System. *Proc. Eighth Symposium on Operating Systems Principles*, p.47-58, Dec. 1981.
- [SV84] SVOBODOVA, L.. File Servers for Network-Based Distributed Systems. *Computing Surveys*, v. 16, n. 4, p. 353-398, Dec. 1984.
- [SU90] SUN MICROSYSTEMS, INC.. SunOS System and Network Administration. California, 1990.
- [TA85] TANENBAUM, A. S. and VAN RENESSE, R.. Distributed Operating Systems. *ACM Computing Surveys*, v. 17, n. 4, p. 419-470, Dec. 1985.
- [TA89] TANENBAUM, A. S.. *Computer Networks*. 2 ed. Englewood Cliffs, Prentice-Hall, 1989.
- [TA87] TANENBAUM, A. S.. *Operating Systems: Design and Implementation*. Englewood Cliffs, Prentice-Hall, 1987.
- [WI91] WITTIE, L. D.. Computer Networks and Distributed Systems. *IEEE Computer*, v. 24, n. 9, p. 67-77, Sep. 1991.

APÊNDICE A

open - original do MINIX

```
#include "lib.h"
PUBLIC int open(name, mode)
char* name;
int mode;
{
    return callm3(FS, OPEN, mode, name);
}
```

open - proposto para o saa-WORM

```
/* open(name,mode) - Abrir arquivo e re-instalar se estiver no WORM */
#include <lib.h>
#include <signal.h>
#include <sys/stat.h>
#include "worm.h"

#define DIA (24L * 60L * 60L)

PUBLIC int open(name, mode)
char* name;
int mode;
{
    char *cp_cmd[3], *ctime(), nome_abs[NOME_MAX_ABS];
    int fa, fd, fw, pid, status, werro;
    struct stat abuf;

    fa = callm3(FS, OPEN, mode, name);
    printf("i/Abrindo arq: %s fa %d \n", name, fa);
    if (fa <= 0) return(fa);
    /*
     * Verifica se o arquivo esta no WORM (desuso)
     */
    fstat(fa, &abuf);
    printf("st_nlink %d st_size %Ld \n", abuf.st_nlink, abuf.st_size);

    if ((abuf.st_nlink > 1) && (abuf.st_size == 0))
    { wnome_inode(DESUZO, abuf.st_ino, nome_abs, NAO_UNICO);
      fw = callm3(FS, OPEN, 0, nome_abs);
      printf("Tentando WORM %s - fw %d \n", nome_abs, fw);
      if (fw <= 0) return(fa);

      /* Re-instala copia do arquivo */
      werro = callm1(FS, CLOSE, fw, 0, 0, NIL_PTR, NIL_PTR, NIL_PTR);
      werro = callm1(FS, CLOSE, fa, 0, 0, NIL_PTR, NIL_PTR, NIL_PTR);

      cp_cmd[0] = "/usr/worm/wrc";
      cp_cmd[1] = name;
      cp_cmd[2] = 0;

      /* "fork" Cria filho. */
      if ((pid = fork()) < 0)
          return(fa = callm3(FS, OPEN, mode, name));

      if (pid == 0)
          execv(cp_cmd[0], cp_cmd);

      /* Desliga interrupcao e espera. */
      signal(SIGINT, SIG_IGN);
      signal(SIGQUIT, SIG_IGN);

      do {
          ;
      } while (wait(&status) != pid);
    }

    return(fa = callm3(FS, OPEN, mode, name));
}
```

close - original do MINIX

```
#include "lib.h"
PUBLIC int close(fd)
int fd;
{
    return callm1(FS, CLOSE, fd, 0, 0, NIL_PTR, NIL_PTR, NIL_PTR);
}
```

close - proposto para o saa-WORM

```
/* close(fd) - Fechar arquivo e informar ao se o arquivo é critico */

PUBLIC int close(fd)
int fd;
{
    char *cp_cmd[4], nome_crit[NOME_MAX_ABS];
    int erro, fa, fb, fw, pid, status, werro;
    struct stat s;
    reg_dir wdir;
    unsigned short inode;

    fstat(fd, &s);
    inode = s.st_ino;

    /* Verifica se este arquivo e' critico */
    fa = callm3(FS, OPEN, LE, WORM_DIR);

    while (read(fw, &wdir, sizeof(wdir)) == sizeof(wdir))
        if (inode == wdir.inode_arq) break;

    erro = callm1(FS, CLOSE, fd, 0, 0, NIL_PTR, NIL_PTR, NIL_PTR);
    if (erro) return erro;
    if (inode != wdir.inode_arq) return erro;

    /* Verifica se copia e' de mesma data */
    fa = callm3(FS, OPEN, LE, wdir.nome_abs);
    fstat(fa, &s);

    if (s.st_mtime > wdir.d_versao) {
        wnome_inode(CRITICO, inode, nome_crit, NAO_UNICO);

        werro = callm1(FS, CLOSE, fa, 0, 0, NIL_PTR, NIL_PTR, NIL_PTR);
        werro = callm1(FS, CLOSE, fb, 0, 0, NIL_PTR, NIL_PTR, NIL_PTR);
        werro = callm1(FS, CLOSE, fw, 0, 0, NIL_PTR, NIL_PTR, NIL_PTR);

        /* Copia de fa para fb */
        cp_cmd[0] = "/usr/bin/cp";
        cp_cmd[1] = wdir.nome_abs;
        cp_cmd[2] = nome_crit;
        cp_cmd[3] = 0;

        /* "fork" Cria filho. */
        if ((pid = fork()) < 0)
            return(erro);
        if (pid == 0)
            execv(cp_cmd[0], cp_cmd);

        /* Desliga interrupcao e espera. */
        signal(SIGINT, SIG_IGN);
        signal(SIGQUIT, SIG_IGN);

        do {
            ;
        } while (wait(&status) != pid);
    }

    return(erro);
}
```

APÊNDICE B

worm.h - constantes e estruturas para saa-WORM

```

/* Definicao de constantes utilizadas no sistema WORM */
#define BARRA '/'
#define BLOCO_WORM 1024
#define BLOC_2_WORM (BLOCO_WORM + BLOCO_WORM)
#define COPIA "/usr/worm/copia"
#define CORRETO 1
#define CRITICO "/usr/worm/critico"
#define DESUSO "/usr/worm/desuso"
#define DIA_EM_SEG (24L * 60L * 60L)
#define DIAS_DESUSO 180
#define DIR_COR '.\0'
#define ERRADO 0
#define GRAVA 1
#define LE 0
#define LE_GR 2
#define NARQS_BUF 8
#define NAO_UNICO 0
#define NB_REG_U sizeof(reg_user)
#define NB_REG_A sizeof(reg_arq)
#define NB_REG_D sizeof(reg_dir)
#define NOME_MAX_ABS 98
#define NOME_MAX 14
#define UNICO 1
#define WCHAVE "WORM\0"
#define WCHAVE_COP "WORMCA"
#define WCHAVE_CRI "WORMIA"
#define WCHAVE_DES "WORMDA"
#define WORM_DIR "/usr/worm/worm.dir"
#define WORM_MID "/usr/worm/worm.mid"
#define WUSERS "/usr/worm/wusers"
/* Estrutura do arquivo de usuarios wuser */
typedef struct {
    int uid,
        n_mb;
    } reg_user;

/* Estrutura do arquivo fila de espera para transferencia */
typedef struct {
    short int dev;
    unsigned short inode_dir;
    unsigned short inode_arq;
    long tempo;
    } reg_arq;

/* Estrutura do arquivo de diretorio do sistema de arquivos WORM */
typedef struct {
    short int dev; /* Dispositivo (Major/Minor) origem arq. */
    unsigned short inode_dir; /* Inode do diretorio do arquivo */
    unsigned short inode_arq; /* Inode do arquivo */
    long tam_arq; /* Tamanho do arquivo em bytes */
    long d_versao; /* Data/hora da ultima versao */
    int nmidia; /* Numero de serie da midia usada */
    long bloc_worm; /* Endereco bloco inicio arquivo */
    int uid; /* Numero do usuario */
    int gid; /* Numero do grupo do usuario */
    char chave[6]; /* Senha para identificacao do cabecalho */
    char nome_abs[NOME_MAX_ABS]; /* Nome absoluto do
arquivo */
    } reg_dir;

/* Estrutura do arquivo de controle de midia do WORM */
typedef struct {
    short int nserie; /* Numero de serie da midia */
    long nblocs; /* Numero total de blocos da midia */
    long ublus; /* Ultimo bloco de dados usado na midia */
    long bloc_dir; /* Bloco de Inicio do diretorio da midia */

```

```

    long ctime; /* Data/hora da inicializacao da midia */
    int dev; /* Dispositivo (Major/Minor) se instalado */
    char id_mid[16]; /* Identificacao da midia */
    int narqs_buf; /* Numero de arquivos atualmente no buffer */
    char buf_dir[BLOCO_WORM]; /* Buffer do diretorio da midia
*/
    } w_mid;

/* Estrutura da tabela de dispositivos possiveis para simulacao do WORM
*/
struct {
    int dev;
    int nblocs;
    char nome[NOME_MAX];
    } dev_tab[3] = {
    {
        01000, 360, "/dev/fd0\0\0\0\0\0",
        01001, 1200, "/dev/at1\0\0\0\0\0",
        00000, 0, "\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"
    }
};

```

procedimentos comuns

```

int acrarq(chave)
char *chave;
/* Acrescentar a identificacao do arquivo na lista do WORM */
{
    int fd, inc;
    long t;
    reg_dir reg;

    fd = open(WORM_DIR, LE_GR);
    if (fd < 0) {
        printf("Falha no sistema de arquivo WORM. Instalacao incompleta. \n");
        exit(ERRADO);
    }

    /* Verifica se este arquivo ja consta do diretorio do WORM */
    inc = 1;
    while (read(fd, &reg, sizeof(reg)) == sizeof(reg)) {
        if ((reg.inode_arq == abuf.st_ino) && (reg.dev == abuf.st_dev))
            { inc = 0;
              break;
            }
    }

    /* Inicializa informacoes do arquivo */
    if (inc) {
        reg.dev = abuf.st_dev;
        reg.inode_dir = dbuf.st_ino;
        reg.inode_arq = abuf.st_ino;
        reg.tam_arq = abuf.st_size;
        if (chave == WCHAVE_DES)
            reg.d_versao = abuf.st_atime;
        else
            reg.d_versao = abuf.st_mtime;
        reg.nmidia = 0;
        reg.bloc_worm = 0L;
    }
    else
        lseek(fd, (long) -sizeof(reg), 1);

    reg.uid = abuf.st_uid;
    reg.gid = abuf.st_gid;
    strcpy(reg.chave, chave);
    strcpy(reg.nome_abs, nome_abs);

    write(fd, &reg, sizeof(reg));
}

```

```

close(fd);
return(CORRETO);
}

int faznomeabs(nome_rel,nome_abs)
char *nome_rel, *nome_abs;
{
    char *d, *p, partes[16][14];
    int dev_cor, dev_pai, fa, i, n;
    struct direct dir;
    struct stat s, ss;
    unsigned short inode_cor, inode_pai;

    /* Verifica se o nome dado e' absoluto */

    if (*nome_rel == '/') {
        strcpy(nome_abs,nome_rel);
        return(CORRETO);
    }

    n=0;
    /* Pega a identificacao do diretorio corrente */
    stat(".",&s);
    do {
        if ((fa = open("../",LE)) <= 0) {
            printf("Nao e' possivel abrir ../\n");
            return(ERADO);
        }
        dev_cor = s.st_dev;
        inode_cor = s.st_ino;

        stat("../",&s);
        dev_pai = s.st_dev;
        inode_pai = s.st_ino;

        chdir("../");

        if (dev_cor == dev_pai)
            do if (read(fa,&dir,sizeof(dir)) < sizeof(dir)) {
                printf("Nao e' possivel ler pai \n");
                return(ERADO);
            }
            while (dir.d_ino != inode_cor);
        else
            do { if (read(fa,&dir,sizeof(dir)) < sizeof(dir)) {
                printf("Nao e' possivel ler ../\n");
                return(ERADO);
            }
                stat(dir.d_name,&ss);
            } while ((ss.st_dev != dev_cor) || (ss.st_ino != inode_cor));

        close(fa);
        if (strcmp(".",dir.d_name)) {
            strcpy(partes[n++],dir.d_name);
            if (n>14) {
                printf("Caminho muito longo. Mais de 15 niveis \n");
                return(ERADO);
            }
        }
    } while ((inode_cor != inode_pai) || (dev_cor != dev_pai));

    p = nome_abs;
    *p++ = '/';

    for (;n>0;) {
        d = partes[--n];
        while (*d) *p++ = *d++;
        *p++ = '/';
    }
    strcpy(p,nome_rel);
    return(CORRETO);
}

```

```

}
int inodedir(nome_rel, nome_abs)
char *nome_rel, *nome_abs;
/* Pegar o inode do diretorio a que pertence o arquivo dado */
{ char *p, *ultc;
  int fd, s;

  strcpy(nome_abs,nome_rel);
  ultc = 0;
  p = nome_abs;

  while (*p) {
      if ( *p == '/')
          ultc = p;
      p++;
  }

  if ( !ultc )
      ultc = ".";
  else
      { *ultc = '\0';
        ultc = nome_abs;
      }

  if (stat(ultc,&dbuf) < 0)
      return(ERADO);
  else
      return(dbuf.st_ino);
}

int permissao()
/*
 * Verifica se o usuario tem permissao de acesso ao WORM
 */
{
    int fd, i, n_users;
    reg_user reg;
    uid user;

    fd = open("/usr/worm/wusers",0);
    if (fd < 0)
        { printf("WORM nao disponivel \n");
          return(ERADO);
        }

    user = getuid();
    read(fd,&reg,NB_REG_U);
    n_users = reg.uid;

    for (i=0;i<n_users;i++)
        { read(fd,&reg,NB_REG_U);
          if (reg.uid == user)
              return(CORRETO);
          }
    return(ERADO);
}

int wnome_inode(dir,inode,abs,opcao)
char *dir;
unsigned short inode;
char *abs;
short opcao;
/*
 * wnome_inode - cria nome para arquivo a partir do i-node
 */
{ char *c, *p, *itoa();
  int n;
  struct stat s;

  /* Cria o nome absoluto do arquivo no diretorio
  ../worm/<critico|copia|desuso> a partir do inode do arquivo */

```

```

c = dir;
p = abs;
while (*p++ = *c++);
*--p = '/';
*++p = 'A';
c = itoa(inode);
while (*++p = *c++);
*++p = '\0';

/* Verifica se a entrada sera' unica no diretorio */
if(opcao == UNICO)
    if ( (n=stat(abs,&s)) >= 0)
        unlink(abs);

return(CORRETO);
}

```

procedimentos principais do saa-WORM

wci

```

/* wci - Acrescentar identificacao do arquivo critico na lista de arquivos a
serem
*   copiados para o WORM.
*/
/*
* wci.c  1.1   10/mar/93
*
* Luiz A. F. Bezerra
*
* Sintaxe:
*
*   wci <id_arquivo>
*
*/

#include <time.h>
#include <h/const.h>
#include <h/type.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include "worm.h"

char *ctime();
char *itoa();
char nome_abs[NOME_MAX_ABS];
char nome_cp[NOME_MAX_ABS];
char *cp_cmd[4];

```

```

int inodendir(), open(), permissao(), read(), strcmp();
long time();
struct stat abuf, dbuf, s;

```

```

main(n_args, l_args)
int n_args;
char *l_args[];
{
    int fd, i_dir_cor, m, s;
    uid user;
    /*
    Verifica se o usuario tem permissao para usar o WORM
    */
    if (permissao() == ERRADO)
    { printf("Acesso nao permitido \n");
      exit(ERRADO);
    }

    if (n_args < 2)

```

```

{ printf("Faltou a identificacao do Arquivo \n");
  exit(ERRADO);
}

fd = open(l_args[1],0);
if (fd < 0)
{ printf("Arquivo nao encontrado ou acesso nao permitido \n");
  exit(ERRADO);
}
s = fstat(fd,&abuf);
m = abuf.st_mode & S_IFMT;
/*
* printf("i_node: %d dev: %o links: %d uid: %d at: %d %d
\n",abuf.st_ino
* ,abuf.st_dev,abuf.st_nlink,abuf.st_uid,abuf.st_mtime);
* printf("at: %s \n",ctime(&abuf.st_mtime));
*/
if (((user = getuid()) != abuf.st_uid) && (user != SUPER_USER))
{ printf("Arquivo nao pertence ao usuario \n");
  exit(ERRADO);
}

if (!(s >= 0 && m == S_IFREG))
{ printf("Arquivo nao e' do tipo regular \n");
  exit(ERRADO);
}

if ((i_dir_cor = inodendir(l_args[1]),nome_abs) == ERRADO)
{ printf("Falha na busca do diretorio do arquivo \n");
  exit(ERRADO);
}

faznomeabs(l_args[1],nome_abs);

if (acrarg(WCHAVE_CRI) == ERRADO)
{ printf("Falha na lista de copia do WORM \n");
  exit(ERRADO);
}

/*
* Garantir unicidade do nome no diretorio ../worm/critico
*/
wnome_inode(CRITICO,abuf.st_ino,nome_cp,UNICO);
cp_cmd[0] = "/usr/bin/cp";
cp_cmd[1] = nome_abs;
cp_cmd[2] = nome_cp;
cp_cmd[3] = 0;

printf("abs: %s worm: %s \n",nome_abs,nome_cp);
execv(cp_cmd[0],cp_cmd);
printf("executou execv \n");
}

```

wcp

```

/* wcp - Acrescentar identificacao do arquivo na lista de arquivos a serem
*   copiados para o WORM */
/*
* wcp.c  1.1   25/fev/93
*
* Luiz A. F. Bezerra
*
* Sintaxe:
*
*   wcp <id_arquivo>
*
*/

#include <time.h>
#include <h/const.h>
#include <h/type.h>

```

```

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include "worm.h"

char *ctime();
char *itoa();
char nome_abs[NOME_MAX_ABS];
char nome_cp[NOME_MAX_ABS];
char *cp_cmd[4];

int inodendir(), open(), permissao(), read(), strcmp();
long time();
struct stat abuf, dbuf;

main(n_args, l_args)
int n_args;
char *l_args[];
{
    int fd, i_dir_cor, m, s;
    uid user;
    /*
    Verifica se o usuario tem permissao para usar o WORM
    */
    if (permissao() == ERRADO)
    { printf("Acesso nao permitido \n");
      exit(ERRADO);
    }

    if (n_args < 2)
    { printf("Faltou a identificacao do Arquivo \n");
      exit(ERRADO);
    }

    fd = open(l_args[1], 0);
    if (fd < 0)
    { printf("Arquivo nao encontrado ou acesso nao permitido \n");
      exit(ERRADO);
    }
    s = fstat(fd, &abuf);
    m = abuf.st_mode & S_IFMT;
    /*
    * printf("i_node: %d dev: %o links: %d uid: %d at: %d %d
    \n", abuf.st_ino
    * , abuf.st_dev, abuf.st_nlink, abuf.st_uid, abuf.st_mtime);
    * printf("at: %s \n", ctime(&abuf.st_mtime));
    */
    if (((user = getuid()) != abuf.st_uid) && (user != SUPER_USER))
    { printf("Arquivo nao pertence ao usuario \n");
      exit(ERRADO);
    }

    if (!(s >= 0 && m == S_IFREG))
    { printf("Arquivo nao e' do tipo regular \n");
      exit(ERRADO);
    }

    if ((i_dir_cor = inodendir(l_args[1]), nome_abs) == ERRADO)
    { printf("Falha na busca do diretorio do arquivo \n");
      exit(ERRADO);
    }

    faznomeabs(l_args[1], nome_abs);

    if (acrarq(WCHAVE_COP) == ERRADO)
    { printf("Falha na lista de copia do WORM \n");
      exit(ERRADO);
    }

    wnome_inode(COPIA, abuf.st_ino, nome_cp, UNICO);
    cp_cmd[0] = "/usr/bin/cp";
    cp_cmd[1] = &nome_abs[0];

```

```

    cp_cmd[2] = nome_cp;
    cp_cmd[3] = 0;
    execv(cp_cmd[0], cp_cmd);
}

```

```

int execute(prg, cp_cmd)
char *prg, **cp_cmd;
{ int i;
  printf("Exec: %s \n", prg);
  i = 0;
  while (*cp_cmd[i]) {
    printf("Arg: %d = %s \n", i, cp_cmd[i]);
    i++;
  }
  return(CORRETO);
}

```

wdes

```

/* wdes - Localiza e cria "link" no diretorio /usr/worm/desuso para
transferencia para o WORM, de todos os arquivos que nao foram
acessados a mais de <n> dias */

```

```

/*
* wdes.c 1.1 27/5/93
*
* Luiz A. F. Bezerra
*
*/

```

```

#include <stdio.h>
#include <h/type.h>
#include <h/const.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "worm.h"

```

```

typedef struct {
    int inum;
    char d_nome[NOME_MAX];
} DIR;

```

```

char *inic_dir = "/"; /* Busca inicial: raiz do sistema MINIX */
char *d_link = "/usr/worm/desuso";
char *itoa();
DIR dir;

```

```

int dev_worm, fd, hoje, ndias;
int open(), read(), stat(), strlen();
long t;
reg_dir reg;
struct stat abuf, dbuf, s;

```

```

main()
{
    time(&t);
    hoje = t/DIA_EM_SEG;
    /*
    Pega identificacao (Major/Minor) de onde esta o controle do WORM
    */
    stat(d_link, &abuf);
    dev_worm = abuf.st_dev;

    fd = open(WORM_DIR, LE_GR);
    if (fd < 0) {
        printf("Falha no sistema de arquivos WORM. Instalacao incompleta. \n");
        exit(ERRADO);
    }
    /*
    * Permitir escolha mais adequada dos arquivos em desuso por <ndias>
    *
    * read(fd, &reg, sizeof(reg));
    */
}

```



```

* DIAS_DESUSO = reg.dev;
*/
le_dir(inic_dir);

close(fd);
exit(0);
}

/*
* Acrescentar a identificacao do arquivo no diretorio do WORM
*/
int acrarq(nome_abs)
char *nome_abs;
{
    int inc;
    long t;

    /* Verifica se este arquivo ja consta do diretorio do WORM */
    inc = 1;
    lseek(fd,0L,0);
    while (read(fd,&reg,sizeof(reg)) == sizeof(reg)) {
        if ((reg.inode_arq == abuf.st_ino) && (reg.dev == abuf.st_dev))
        {
            inc = 0;
            break;
        }
    }

    /* Inicializa informacoes do arquivo */
    if (inc) {
        reg.dev = abuf.st_dev;
        reg.inode_dir = dbuf.st_ino;
        reg.inode_arq = abuf.st_ino;
        reg.tam_arq = abuf.st_size;
        reg.d_versao = abuf.st_atime;
        reg.nmidia = 0;
        reg.bloc_worm = 0L;
    }
    else
        lseek(fd,(long) -sizeof(reg),1);

    reg.uid = abuf.st_uid;
    reg.gid = abuf.st_gid;
    strcpy(reg.chave,WCHAVE_DES);
    strcpy(reg.nome_abs,nome_abs);

    write(fd,&reg,sizeof(reg));

    return(CORRETO);
}

/*
* le_dir - Percorre todo o diretorio dado e seus descendentes, procurando
* arquivos regulares nao acessados ha mais de DIAS_DESUSO
*/
le_dir(d)
char *d;
{
    char nome_abs[NOME_MAX_ABS];
    char nome_ln[NOME_MAX_ABS];
    int fd;

    if ((fd = open(d,0)) < 0)
        return(0);
    /*
    * Escolhe somente arquivos regulares
    */
    while (read(fd, &dir, sizeof(dir)) > 0) {
        if (strcmp(dir.d_nome, ".") == 0 || strcmp(dir.d_nome, "..") == 0)
            continue;
        if (dir.inum == 0)
            continue;
        if (!pac_nome(d, dir.d_nome, nome_abs))

```

```

            continue;
        if (stat(nome_abs, &abuf) < 0)
            continue;
        if (abuf.st_dev != dev_worm)
            continue;
        if ((abuf.st_mode & S_IFMT) == S_IFDIR)
            le_dir(nome_abs);
        stat(nome_abs,&abuf); /* E' necessario por causa da recursao ! */
        if ((abuf.st_mode & S_IFMT) != S_IFREG)
            continue;
        if (abuf.st_nlink > 1)
            continue;
        {
            ndias = hoje - (abuf.st_atime/DIA_EM_SEG);
            printf("arq: %s ndias %d size %d\n",nome_abs,ndias,abuf.st_size);
            if ((ndias > DIAS_DESUSO) && (abuf.st_size > BLOC_2_WORM))
            {
                stat(d,&dbuf);
            }
        }
        /*
        * Acrescenta a identificacao do arquivo no diretorio do WORM
        */
        acrarq(nome_abs);
        /*
        * Garante nome unico na fila de arquivos a transferir para o WORM
        * (/usr/worm/desuso/)
        */
        wnome_inode(d_link,abuf.st_ino,nome_ln,NAO_UNICO);
        /*
        * Cria entrada no diretorio ../worm/desuso ligada ao arquivo em
        questao
        */
        printf("%s <- %s\n",nome_abs,nome_ln);
        link(nome_abs,nome_ln);
    }
    continue;
}
close(fd);
return(CORRETO);
}

/*
* pac_nome - cria o nome absoluto (completo) do arquivo
*/
int pac_nome(dir, arq, abs)
char *dir;
char *arq;
char *abs;
{
    char *cp;
    int n;
    n = MIN(strlen(arq),14);

    if (strlen(dir) + n + 2 > NOME_MAX_ABS)
        return(0);
    for (cp = abs ; *dir ; *cp++ = *dir++)
        ;
    if (*(cp-1) != '/')
        *cp++ = '/';
    while (n--)
        *cp++ = *arq++;
    *cp = '\0';

    return(1);
}

```

wdm

```

/* wdm - Desmonta midia em uso no sistema WORM
*
* wdm <dispositivo>
*
* dispositivo - identificacao do dispositivo no MINIX.
*

```



```

w_mid wmid;

struct stat s;

main()
{
    int status, servico(), limpa_f();

    /* Coloca wgt em background "forking" e termina execucao do pai. */

    pid = getpid();
    /* printf("pid.antes.fork: %d\n",pid); */

    status=fork();

    if ( status == (-1) ) /* Erro - nao e possivel fork */
    { printf("wgt: Erro - nao e possivel fork!\n");
      exit(-1);
    }

    pid = getpid();
    /* printf("pid.apos.fork: %d\n",pid); */

    if (status > 0) exit(0); /* pai termina */

    signal(SIGINT, SIG_IGN); /* interrupt (DEL) */
    signal(SIGHUP, SIG_IGN); /* hangup */
    signal(SIGQUIT, SIG_IGN); /* quit (ASCII FS) */

    /* close(0); close(1); close(2); */ /* * fecha todos I/O paths */

    for (;;)
    { signal(SIGALRM, servico); /* Indica procedimento de transferencia */

      pid = getpid();
      /* printf("pid.antes.alarm: %d\n",pid); */

      alarm(5); /* Programa reativacao do programa */

      pid = getpid();
      /* printf("pid.apos.alarm: %d\n",pid); */

      pause(); /* para cada 30 segundos */

      pid = getpid();
      /* printf("pid.apos.pause: %d\n",pid); */

      signal(SIGALRM, limpa_f);
      alarm(1);

      pid = getpid();
      /* printf("pid.antes.wait: %d\n",pid); */

      wait((int *)NULL); /* limpa filhos zumbis */

      pid = getpid();
      /* printf("pid.apos.wait: %d\n",pid); */
    }
}

int limpa_f()
{
    pid = getpid();
    /* printf("pid.na.limpa_f: %d\n",pid); */
    /* Restaurar a integridade do sistema (Gerente de Memoria). */
}

servico()
{
    /* Verifica o que existe em cada um dos diretorios */

    pid = getpid();

    fi = open(CRITICO,LE);

    printf("Fila %s fd %d\n",CRITICO,fi);

    /* Arquivos Criticos */
    le_dir(fi,CRITICO,REMOVE);
    close(fi);

    /* Copia de Seguranca */
    fc = open(COPIA,LE);
    printf("Fila %s fd %d\n",COPIA,fc);
    le_dir(fc,COPIA,REMOVE);
    close(fc);

    /* Arquivos em desuso */
    fd = open(DESUSO,LE);
    printf("Fila %s fd %d\n",DESUSO,fd);
    le_dir(fd,DESUSO,MANTEM);
    close(fd);
}

char *dev_nome(dev)
int dev;
{ int i;
  for (i=0; dev_tab[i].dev; i++)
    if (dev_tab[i].dev == dev) break;
  return(dev_tab[i].nome);
}

/*
 * le_dir - processa todo o directorio aberto
 */
le_dir(fd,id_dir,remove)
char *id_dir;
int fd, remove;
{ char id_arq[NOME_MAX_ABS];
  int erro;

  lseek(fd,0L,0);
  while (read(fd, &dir, sizeof(dir)) > 0) {
    if (strcmp(dir.nome_a, ".") == 0 || strcmp(dir.nome_a, "..") == 0)
      continue;
    if (dir.inum_a == 0)
      continue;
    dir_arq(id_dir,dir.nome_a,id_arq);
    if (remove) printf("Abrindo arq %s\n",id_arq);
    erro = stat(id_arq,&s);
    fa = open(id_arq,LE);
    if (remove) printf("Proc %s m %o S_IFREG %o erro %d fa %d s %s%Ld\n",
                      id_arq, s.st_mode, S_IFREG, erro, fa, s.st_size);

    /* if ((fa = open(id_arq,LE)) <= 0) */
    if (fa <= 0)
      continue;
    if ( erro < 0)
      continue;
    if ((s.st_mode & S_IFMT) == S_IFDIR) {
      close(fa);
      continue;
    }
    if (s.st_size == 0) {
      close(fa);
      continue;
    }
    if ((s.st_mode & S_IFMT) == S_IFREG)

```

```

{ if ((erro = salva_worm(id_arq)) < 0) {
    close(fa);
    printf("Erro# %d no WORM. Sistema WORM desativado \n", erro);
    /* Prepara suicidio: kill <proprio pid> */
    exit(ERRADO);
}
if (remove)
{ printf("rm: %s \n", id_arq);
  close(fa);
  unlink(id_arq);
}
else
{ close(fa);
  printf("ln: %s fa %d ", id_arq, fa);
  fa = creat(id_arq, s.st_mode & 0777);
  printf("fa apos: %d erro %d \n", fa, erro);
  close(fa);
}
}
return(1);
}

dir_arq(dir, arq, abs)
char *dir, *arq, *abs;
{ char *d, *p;
  p = abs;
  for (d = dir; *p++ = *d; d++);
  *--p = '/';
  for (d = arq; *++p = *d; d++);
  return;
}

/*
 * loc_arq - localiza o ultimo componente (nome do arquivo) de um path
 */
char *loc_arq(nome_abs)
char *nome_abs;
{
  char *c, *p;
  c = nome_abs;
  p = 0;
  while (*c) {
    if (*c == '/') p = c;
    c++;
  }
  p++;
  return(p);
}

salva_worm(id_arq)
char *id_arq;
{ char c, *loc_arq(), nome_arq[NOME_MAX_ABS], *p;
  int fm, i, livre, nb, mid;
  long t;
  unsigned short inode;
  printf("Salva: %s ", id_arq);
  /* localiza arquivo a ser salvo no WORM atraves do seu inode
   salva arquivo no WORM
   atualiza ultimo bloco usado
   atualiza versao da copia no WORM
  */
  p = loc_arq(id_arq);
  inode = 0;
  while (c = *p++) {
    if ((c < '0') || (c > '9'))
      return(ERRADO);
    inode = inode * 10 + (c - '0');
  }
  printf("inode %d \n", inode);
  t = le_wdir(inode);

```

```

printf("d_versao %8Ld size %d \n", t.s.st_size);
if (t != 0) {
  mid = le_wmid(0, s.st_size);
  printf("Midia escolhida %d \n", mid);
  if (mid <= 0) return(ERRADO);
  p = dev_nome(wmid.dev);
  fm = open(p, LE_GR);
  printf("Dev_nome: %s fm %d \n", p, fm);
  if (fm <= 0) return(ERRADO);

  wdir.bloc_worm = wmid.ublus + 1;
  wdir.nmidia = wmid.nserie;
  lseek(fm, (wdir.bloc_worm)*BLOCO_WORM, 0);

  while ((nb = read(fa, bloco, BLOCO_WORM)) == BLOCO_WORM) {
    if (write(fm, bloco, BLOCO_WORM) != BLOCO_WORM) {
      printf("Falha na midia# %d \n", wmid.nserie);
      return(ERRADO);
    }
    wmid.ublus++;
    printf("%s gravando bloco# %6Ld #bytes %d
\n", up_line, wmid.ublus, nb);
  }

  /* Acrescenta a identificacao do arquivo no final do ultimo bloco */
  if (nb <= 0) nb = 0;
  if ((BLOCO_WORM - nb) < sizeof(wdir)) {
    if (write(fm, bloco, BLOCO_WORM) != BLOCO_WORM) {
      printf("Falha na midia# %d \n", wmid.nserie);
      return(ERRADO);
    }
    wmid.ublus++;
    printf("%s gravando bloco# %6Ld #bytes %d
\n", up_line, wmid.ublus, nb);
    nb = 0;
  }
  for (i = nb; i < BLOCO_WORM; bloco[i++] = 0);
  bcopy(&wdir, &bloco[BLOCO_WORM - sizeof(wdir)], sizeof(wdir));
  if (write(fm, bloco, BLOCO_WORM) != BLOCO_WORM) {
    printf("Falha na midia# %d \n", wmid.nserie);
    return(ERRADO);
  }
  wmid.ublus++;
  printf("%s gravando bloco# %6Ld #bytes %d
\n", up_line, wmid.ublus, nb);

  /* Atualiza o diretorio da midia */
  if (wmid.narqs_buf == NARQS_BUF) {
    wmid.bloc_dir--;
    printf("Gravando bloc_dir# %6Ld \n", wmid.bloc_dir);
    lseek(fm, wmid.bloc_dir*BLOCO_WORM, 0);
    write(fm, wmid.buf_dir, BLOCO_WORM);
    wmid.narqs_buf = 0;
    for (i = 0; i < BLOCO_WORM; wmid.buf_dir[i++] = 0);
  }

  bcopy(&wdir, &wmid.buf_dir[wmid.narqs_buf*sizeof(wdir)], sizeof(wdir));
  wmid.narqs_buf++;

  gr_wmid(wmid.nserie);
  gr_wdir(wdir.inode_arq);

  return(CORRETO);
}
else
  return(ERRADO);
}

long gr_wdir(inode)
unsigned short inode;
{ int fw;
  reg_dir zdir;

```

```

fw = open(WORM_DIR,LE_GR);
while (read(fw,&zdir,sizeof(zdir)) == sizeof(zdir)) {
    if (inode == zdir.inode_arq) {
        lseek(fw,(long)(-sizeof(zdir)),1);
        write(fw,&wdir,sizeof(wdir));
        close(fw);
        return(wdir.d_versao);
    }
}
close(fw);
return(ERADO);
}

int gr_wmid(n)
int n;
{ int fm, nb;
  w_mid zmid;
  fm = open(WORM_MID,LE_GR);
  read(fm,&zmid,sizeof(zmid));
  if (zmid.nserie < n) {
      close(fm);
      return(ERADO);
  }
  do {
      if ((nb = read(fm,&zmid,sizeof(zmid))) == sizeof(zmid))
          if (n == zmid.nserie) {
              lseek(fm,(long)(-sizeof(zmid)),1);
              write(fm,&wmid,sizeof(wmid));
              close(fm);
              return(zmid.dev);
          }
  } while (nb > 0);
  return(ERADO);
}

long le_wdir(inode)
unsigned short inode;
{ int fw;
  fw = open(WORM_DIR,LE);
  printf("abrindo %s fw %d\n",WORM_DIR,fw);
  while (read(fw,&wdir,sizeof(wdir)) == sizeof(wdir)) {
      printf("proc inode %d %d\n",inode,wdir.inode_arq);
      if (inode == wdir.inode_arq) {
          close(fw);
          return(wdir.d_versao);
      }
  }
  close(fw);
  return(ERADO);
}

int le_wmid(n,area)
int n;
long area;
{ int fm, nb;
  int i;
  fm = open(WORM_MID,LE);
  read(fm,&wmid,sizeof(wmid));
  if (wmid.nserie < n) {
      close(fm);
      return(ERADO);
  }
  do {
      if ((nb = read(fm,&wmid,sizeof(wmid))) == sizeof(wmid))
          if ((n == 0) || (n == wmid.nserie) &&
              ((wmid.bloc_dir - wmid.ublus)*BLOCO_WORM >= area)) {
              close(fm);
              return(wmid.dev);
          }
  } while (nb > 0);
  return(-n);
}

```

wim

```

/* wim - Inicializa midia para uso no sistema WORM
 *
 * wim <dispositivo> <tamanho> <rotulo>
 *
 * dispositivo - identificacao do dispositivo no MINIX.
 *
 * tamanho    - numero total de blocos do dispositivo
 *
 * rotulo      - identificacao para a midia (ate 16 caracteres)
 *
 * ex.: wim /dev/at1 1200 disq001
 *
 * Luiz A. F. Bezerra
 */

#include <sys/stat.h>
#include "worm.h"

char *dispositivo;
char bloco[BLOCO_WORM];
int open(), strcmp(), write();
long atol(), d, lseek(), time();

struct stat dbuf;

reg_dir w_arq;
w_mid w, wi;

main(nargs,largs)
int nargs;
char *largs[];
{
    char *up_line = "\033[A\0";
    char *c, *de, *para, resp[1];
    int fd, fm, i, k, erro, s;
    long t;

    if (nargs != 4) {
        uso("Faltou parametro: Uso wim <dev> [<#blocos> <ident.midia>]\n");
        exit(ERADO);
    }
    /*
     * Verifica se o dispositivo e' valido
     */
    dispositivo = largs[1];
    fd = open(dispositivo,LE_GR);
    if (fd <= 0) {
        printf("Dispositivo %s invalido. \n",dispositivo);
        exit(ERADO);
    }
    s = fstat(fd,&dbuf);

    /* Pega o ultimo numero de serie usado */
    while ((fm = open(WORM_MID,LE_GR)) <= 0) {
        printf("Sistema WORM inconsistente. Falta arquivo controle de midia. \n");
        printf("Deseja cria-lo? <s/n>:");
        gets(resp);
        if ((resp[0] != 's') && (resp[0] != 'S')) {
            printf("Inicializacao cancelada. \n");
            exit(ERADO);
        }
        fm = creat(WORM_MID,0644);
        if (fm <= 0) {
            printf("Nao foi possivel criar o arquivo controle de midia. \n");
            exit(ERADO);
        }
        w.nserie = 0;
    }
}

```

```

w.nblocs = 0;
w.ublus = -1L;
w.bloc_dir = 0;
w.ctime = time(&t);
w.dev = 0;
w.narqs_buf = 0;
for (i=0; i<BLOCO_WORM; w.buf_dir[i++]=0);
write(fm,&w,sizeof(w));
close(fm);
}
/* Pega proximo numero de serie para a midia */
read(fm,&w,sizeof(w));
w.nserie++;

w.nblocs = atol(largs[2]);
if ((w.nblocs < 360) || (w.nblocs > 1200)) {
    printf("Numero de blocos %d < 360 ou > 1200!? \n",w.nblocs);
    exit(ERRADO);
}
/* Inicializa parametros de controle da midia */
w.ublus = -1L;
strcpy(w.id_mid,largs[3]);
w.ctime = time(&t);
w.dev = dbuf.st_rdev;
w.bloc_dir = w.nblocs-1;
w.narqs_buf = 0;

/* Verifica se o dispositivo esta em uso */
d = sizeof(wi);
lseek(fm,d,0);
while(read(fm,&wi,sizeof(wi)) == sizeof(wi)) {
    if (w.dev == wi.dev) {
        if (wi.nserie > 0) {
            printf("Dispositivo ocupado. Inicializacao cancelada. \n");
            exit(ERRADO);
        }
        else {
            lseek(fm,(long)-sizeof(wi),1);
            break;
        }
    }
}
for (i=0; i<BLOCO_WORM; w.buf_dir[i++]=0);

if (read(fd,bloco,BLOCO_WORM) > 0) {
    printf("Esta midia nao esta vazia. Deseja usa-la mesmo assim? <s/n>:");
    gets(resp);
    if ((resp[0] != 's') && (resp[0] != 'S')) {
        printf("Inicializacao cancelada! \n");
        exit(ERRADO);
    }
}

lseek(fd,0L,0);

/* Inicializacao especifica para simulacao do WORM a fim de possibilitar
a localizacao dos blocos ja gravados */
printf("\n");
for (i=0; i < BLOCO_WORM; i++) bloco[i] = (char) 0255;
for (i=0; i < w.nblocs; i++) {
    erro = write(fd,bloco,BLOCO_WORM);
    printf("%s gravando bloco# %d #bytes: %d \n",up_line,i,erro);
}

/* Posiciona no final da midia e grava identificacao */
d = (w.nblocs-1)*BLOCO_WORM; /* poderia ser:
    * lseek(fm,(long)-BLOCO_WORM,2)
    * mas nao funciona ?! */

lseek(fd,d,0);
w_arq.tam_arq = w.nblocs;
w_arq.nmidia = w.nserie;
w_arq.d_versao = time(&t);

```

```

strcpy(w_arq.chave,WORM_CHAVE);
strcpy(w_arq.nome_abs,w.id_mid);
bcopy(&w_arq,&bloco[0],sizeof(w_arq));

for (para = &bloco[sizeof(w_arq)]; para < &bloco[BLOCO_WORM-1];
para++) *para = 0;

erro = write(fd,bloco,BLOCO_WORM);
close(fd);

/* Monta a midia para uso */
write(fm,&w,sizeof(w));

/* Atualiza o numero de midias */
lseek(fm,0L,0);
read(fm,&w,sizeof(w));
w.nserie++;
lseek(fm,0L,0);
write(fm,&w,sizeof(w));
close(fm);

fm = open(WORM_MID,LE);
while (read(fm,&w,sizeof(w)) == sizeof(w)) {
    printf("serie %5d #b %6Ld ult.b %6Ld b.dir %6Ld creat %9Ld dev %0
\n",
        w.nserie,w.nblocs,w.ublus,w.bloc_dir,w.ctime,w.dev);
}

fd = open(dispositivo,LE);
printf("midia instalada %s - fd %d \n",dispositivo,fd);
lseek(fd,(w.nblocs-1)*BLOCO_WORM,0);
i = read(fd,bloco,BLOCO_WORM);

bcopy(&bloco[0],&w_arq,sizeof(w_arq));
printf("serie %5d #b %6Ld creat %9Ld chave %s nome %s \n",

w_arq.nmidia,w_arq.tam_arq,w_arq.d_versao,w_arq.chave,w_arq.nome_abs
);

close(fd);
close(fm);
}

bcopy(fnt,dst,nb)
register char *fnt, *dst;
register short nb;
{
    while (nb--) *dst++ = *fnt++;
}

uso(msg)
char *msg;
{ printf("Erro. %s -> Uso: wim <dispositivo> <tamanho> <rotulo>
\n",msg);
    return;
}

int dev_area(dev)
int dev;
{ int i;
    for (i=0; dev_tab[i].dev; i++)
        if (dev_tab[i].dev == dev) break;
    return(dev_tab[i].nblocs);
}

wld

/* wld - Lista o directorio atual do sistema WORM
*
*/

```

```

* wld.c 1.1 10/mar/93
*
* Luiz A. F. Bezerra
*
* Sintaxe:
*
* wld <masc_arquivo>
*
*/

#include <time.h>
#include <h/const.h>
#include <h/type.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include "worm.h"

char *ctime();

int open(), permissao(), read();
long time();

main(n_args, l_args)
int n_args;
char *l_args[];
{
    int fd, i_dir_cor, m, s;
    uid user;
    /*
    Verifica se o usuario tem permissao para usar o WORM
    */
    if (permissao() == ERRADO)
    { printf("Acesso nao permitido \n");
      exit(ERRADO);
    }

    if (lstarq() == ERRADO)
    { printf("Falha na lista do sistema WORM \n");
      exit(ERRADO);
    }
}

int lstarq()
/* Listar a identificacao do arquivo do sistema WORM (diretorio) */
{
    int fd, n;
    long t, tot;
    reg_dir reg;

    fd = open(WORM_DIR, LE);
    if (fd < 0) {
        printf("Falha no sistema de arquivo WORM. Instalacao incompleta. \n");
        exit(ERRADO);
    }
    n = 0; tot = 0;
    while (read(fd, &reg, sizeof(reg)) == sizeof(reg)) {
        n++; tot += reg.tam_arq;
        printf("%5d %5d %6Ld %8Ld ", reg.nmidia, reg.inode_arq, reg.bloc_worm,
              reg.tam_arq);
        date(reg.d_versao);
        printf("%s \n", reg.nome_abs);
    }

    printf("%5d arquivos no sistema WORM. Area total %Ld \n", n, tot);
    close(fd);
    return(CORRETO);
}

```

wlw

```

/* wlw - Lista a situacao atual do sistema de arquivos WORM
*
*
* Luiz A. F. Bezerra
*
* Sintaxe:
*
* wlw [<dispositivo>]
*
*/

#include <time.h>
#include <h/const.h>
#include <h/type.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include "worm.h"

char bloco[BLOCO_WORM], *ctime();
char *up_line = "\033[A\0";
char *dev, *dev_nome();

int fd, fm, open(), permissao(), read(), strcmp();
long d, time();
w_mid w;

main(n_args, l_args)
int n_args;
char *l_args[];
{
    int i_dir_cor, m, s;
    uid user;
    /*
    Verifica se o usuario tem permissao para usar o WORM
    */
    if (permissao() == ERRADO)
    { printf("Acesso nao permitido \n");
      exit(ERRADO);
    }

    if ((fm = open(WORM_MID, LE)) <= 0) {
        printf("Falha no sistema WORM \n");
        exit(ERRADO);
    }
    read(fm, &w, sizeof(w));
    printf("Total de midias inicializadas: %d \n", w.nserie);

    while (read(fm, &w, sizeof(w)) == sizeof(w))
    {
        if (w.nserie > 0) {
            dev = dev_nome(w.dev);
            printf(" %5d %6Ld %6Ld %6Ld %4d %s \n", w.nserie, w.nblocs, w.ublus,
                  w.bloc_dir, w.narqs_buf, dev);
            lstmid(dev, w.nblocs);
        }
    }

    int lstmid(dev, nblocs)
    /* Listar a identificacao do arquivo do sistema WORM (diretorio) */
    char *dev;
    long nblocs;
    { char chave[6];
      int fd, i, n;
      long d;
      reg_dir reg;

      fd = open(dev, LE);
      if (fd < 0) {
          printf("Falha na abertura do dispositivo %s. \n", dev);

```

```

    exit(ERRADO);
}
n = 0;

for (d=0; d<nblocs; d++) {
    /* Verifica se atingiu a area livre da midia */
    if (read(fd,bloco,BLOCO_WORM) != BLOCO_WORM) break;
    if (bloco[0] == (char) 0255) {
        for (i=1; i<BLOCO_WORM; i++)
            if (bloco[i] != (char) 0255) break;
        if (i == BLOCO_WORM) {
            d--;
            break;
        }
    }
    bcopy(&bloco[BLOCO_WORM - sizeof(reg.nome_abs) -
sizeof(reg.chave)]
,&chave[0],5);
    chave[4] = '/';
    chave[5] = '\0';
    if (strcmp(chave,WORM_CHAVE) == 0) {
        n++;
        bcopy(&bloco[BLOCO_WORM - sizeof(reg)],&reg,sizeof(reg));
        printf("%5d %6Ld %8Ld ",reg.nmidia,reg.bloc_worm,reg.tam_arq);
        date(reg.d_versao);
        printf("%s\n",reg.nome_abs);
    }
}

printf("%5d arquivos no sistema WORM - Total blocos usados %6Ld\n",n,d);

/* lseek(fd,(long)-BLOCO_WORM,1);
while (read(fd,bloco,BLOCO_WORM) == BLOCO_WORM);
*/

close(fd);
return(CORRETO);
}

```

```

char *dev_nome(dev)
int dev;
{ int i;
  for (i=0; dev_tab[i].dev; i++)
    if (dev_tab[i].dev == dev) break;
  return(dev_tab[i].nome);
}

```

wmm

```

/* wmm - Monta midia para uso no sistema WORM
*
* wmm <dispositivo> <#blocos>
*
* dispositivo - identificacao do dispositivo no MINIX.
*
* #blocos - numero total de blocos da midia
*
* ex.: wmm /dev/at1 1200
*
* Luiz A. F. Bezerra
*
*/

```

```

#include <sys/stat.h>
#include "worm.h"

```

```

char *dispositivo;
char bloco[BLOCO_WORM];
int open(), strcmp(), write();

```

```

long atol(), d, lseek(), time();

```

```

struct stat dbuf;

```

```

reg_dir w_arq;
w_mid w, wi;

```

```

main(nargs,largs)

```

```

int nargs;

```

```

char *largs[];

```

```

{
    char *up_line = "\033[A\0";
    char *c, *de, *para, resp[1];
    int fd, fm, i, k, erro, s;
    long nblocs, t;

```

```

    if (nargs != 3) {
        uso("Faltou parametro.");
        exit(ERRADO);
    }

```

```

    /*
    * Verifica se o dispositivo e' valido
    */

```

```

    dispositivo = largs[1];
    fd = open(dispositivo,LE_GR);
    if (fd <= 0) {
        printf("Dispositivo %s invalido. \n",dispositivo);
        exit(ERRADO);
    }
    s = fstat(fd,&dbuf);

```

```

    /* Verifica se o dispositivo esta livre */
    if((fm = open(WORM_MID,LE_GR)) <= 0) {
        printf("Sistema WORM inconsistente. Falta arquivo controle de midia.
\n");
        exit(ERRADO);
    }

```

```

    w.dev = dbuf.st_rdev;
    nblocs = atol(largs[2]);
    if (!(nblocs == 360 || nblocs == 1200)) {
        printf("Numero de blocos %d < 360 ou > 1200. \n",nblocs);
        exit(ERRADO);
    }

```

```

    d = sizeof(wi);
    lseek(fm,d,0);
    while(read(fm,&wi,sizeof(wi)) == sizeof(wi)) {
        if (w.dev == wi.dev) {
            if (wi.nserie > 0) {
                printf("Dispositivo %s ocupado com a midia# %d. \n",dispositivo
,wi.nserie);
                exit(ERRADO);
            }
            else {
                lseek(fm,(long)-sizeof(wi),1);
                break;
            }
        }
    }
}

```

```

/* Localiza ultimo bloco gravado */
for (d=0; d<nblocs; d++) {
    erro=read(fd,bloco,BLOCO_WORM);
    for (k=0; k<BLOCO_WORM; k++)
        if (bloco[k] != (char) 0255) break;
    if (k == BLOCO_WORM) break;
}
w.ublus = --d;

```

```

/* Localiza inicio do directorio */
for (d=(nblocs-2); d>w.ublus; d--) {
    lseek(fd,d*BLOCO_WORM,0);
    erro=read(fd,bloco,BLOCO_WORM);
}

```



```

    if (bloco[0] == (char) 0255) break;
}
w.bloc_dir = ++d;

/* Posiciona no final da midia e le a identificacao */
d = (nblocs-1)*BLOCO_WORM; /* poderia ser:
    * lseek(fm,(long)-BLOCO_WORM,2)
    * mas nao funciona ?! */
lseek(fd,d,0);
erro = read(fd,bloco,BLOCO_WORM);
bcopy(&bloco[0],&w_arq,sizeof(w_arq));

w.nblocs = w_arq.tam_arq;
w.nserie = w_arq.nmidia;
strcpy(w.id_mid,w_arq.nome_abs);
w.narqs_buf = 0;
for (i=0; i<BLOCO_WORM; w.buf_dir[i++]=0);
printf("midia %5d #b %6Ld ult.b %6Ld b.dir %9Ld dev %o \n",
    w.nserie,w.nblocs,w.ublus,w.bloc_dir,w.dev);

/* Monta a midia para uso */
write(fm,&w,sizeof(w));

close(fd);
close(fm);

fm = open(WORM_MID,LE);
while (read(fm,&w,sizeof(w)) == sizeof(w)) {
    printf("wmid %5d #b %6Ld ult.b %6Ld b.dir %9Ld dev %o \n",
        w.nserie,w.nblocs,w.ublus,w.bloc_dir,w.dev);
}

fd = open(dispositivo,LE);
printf("midia instalada %s - fd %d \n",dispositivo,fd);
lseek(fd,(nblocs-1)*BLOCO_WORM,0);
i = read(fd,bloco,BLOCO_WORM);

bcopy(&bloco[0],&w_arq,sizeof(w_arq));
printf("midia %5d #b %6Ld creat %9Ld chave %s nome %s \n",
    w_arq.nmidia,w_arq.tam_arq,w_arq.d_versao,w_arq.chave,w_arq.nome_abs);

close(fd);
close(fm);
}

bcopy(fnt,dst,nb)
register char *fnt, *dst;
register short nb;
{
    while (nb--) *dst++ = *fnt++;
}

uso(msg)
char *msg;
{
    printf("Erro. %s -> Uso: wmm <dispositivo> <tamanho> \n",msg);
    return;
}

int dev_area(dev)
int dev;
{
    int i;
    for (i=0; dev_tab[i].dev; i++)
        if (dev_tab[i].dev == dev) break;
    return(dev_tab[i].nblocs);
}

```

WRC

/* wrc - Re-instalar copia de arquivo se estiver no WORM

```

*
*
* wrc <arquivo>
*
* <arquivo> - identificacao original do arquivo a ser re-instalado
*/

#include <lib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#include "worm.h"

char nome_abs[NOME_MAX_ABS], *dev_nome(), *pp;
int dev_num, fa, fd, fw, read();

reg_dir wdir;
w_mid wmid;

main(nargs,largs)
int nargs;
char *largs[];
{
    int inst;
    if (nargs < 2) {
        printf("Falta parametro. Uso wrc <arquivo>");
        exit(ERRADO);
    }
    faznomeabs(largs[1],nome_abs);
    printf("Re-instalando arq %s -> %s \n",largs[1],nome_abs);

    fw = open(WORM_DIR,LE);
    inst = 0;
    while (read(fw,&wdir,sizeof(wdir)) == sizeof(wdir)) {
        if (strcmp(nome_abs,wdir.nome_abs) == 0) {
            reinstw(nome_abs);
            inst++;
            break;
        }
    }
    close(fw);
    if (!inst) {
        printf("Nao existe copia de %s no WORM \n",nome_abs);
        exit(ERRADO);
    }
}

int reinstw(nome_abs)
char *nome_abs;
{
    int fd;
    long p;
    reg_dir worm;

    dev_num = le_wmid(wdir.nmidia,0);
    if (dev_num <= 0) {
        printf("O arquivo nao esta disponivel. Re-instala midia# %d \n",
            wdir.nmidia);
        return(ERRADO);
    }
    pp = dev_nome(dev_num);
    fd = open(pp,LE);

    printf("O arquivo encontra-se no dispositivo %s - fd %d \n",pp,fd);
    if (fd <= 0) {
        printf("Falha na abertura do dispositivo %s \n",pp);
        return(ERRADO);
    }
    p = wdir.bloc_worm*BLOCO_WORM;
    lseek(fd,p,0);
}

```

```

fa = open(nome_abs, GRAVA);
if (fa <= 0) {
    fa = creat(nome_abs, 0644);
    if (fa <= 0) {
        printf("Falha na abertura do arquivo %s\n", nome_abs);
        return(ERRO);
    }
}
copyfile(fd, fa, wdir.tam_arq);
return(CORRETO);
}

#define NBBUF 8192
char cdbuf[NBBUF];

copyfile(fd1, fd2, size)
int fd1, fd2;
long size;
{
    int n, m, mode;
    long t;
    struct stat sbuf;

    t = 0;
    do {
        n = read(fd1, cdbuf, NBBUF);
        if (n <= 0) {
            printf("Erro de leitura no WORM. Midia# %d\n", wdir.nmidia);
            return(ERRO);
        }
        t += n;
        if (t > size)
            n = NBBUF + size - t;
        if (write(fd2, cdbuf, n) != n) {
            printf("Falha de gravacao do arquivo de instalacao\n");
            return(ERRO);
        }
    } while (t < size);
    close(fd1);
    close(fd2);
}

int faznomeabs(nome_rel, nome_abs)
char *nome_rel, *nome_abs;
{
    char *d, *p, partes[16][14];
    int dev_cor, dev_pai, fa, i, n;
    struct direct dir;
    struct stat s, ss;
    unsigned short inode_cor, inode_pai;

    /* Verifica se o nome dado e' absoluto */

    if (*nome_rel == '/') {
        strcpy(nome_abs, nome_rel);
        return(CORRETO);
    }

    n=0;
    /* Pega a identificacao do directorio corrente */
    stat(".", &s);
    do {
        if ((fa = open(".", LE)) <= 0) {
            printf("Nao e' possivel abrir ..\n");
            return(ERRO);
        }
        dev_cor = s.st_dev;
        inode_cor = s.st_ino;

        stat("..", &s);
        dev_pai = s.st_dev;

```

```

        inode_pai = s.st_ino;

        chdir("..");

        if (dev_cor == dev_pai)
            do if (read(fa, &dir, sizeof(dir)) < sizeof(dir)) {
                printf("Nao e' possivel ler pai\n");
                return(ERRO);
            }
            while (dir.d_ino != inode_cor);
        else
            do { if (read(fa, &dir, sizeof(dir)) < sizeof(dir)) {
                printf("Nao e' possivel ler ..\n");
                return(ERRO);
            }
            stat(dir.d_name, &ss);
        } while ((ss.st_dev != dev_cor) || (ss.st_ino != inode_cor));

        close(fa);
        if (strcmp(".", dir.d_name)) {
            strcpy(partes[n++], dir.d_name);
            if (n > 14) {
                printf("Caminho muito longo. Mais de 15 niveis\n");
                return(ERRO);
            }
        }
    } while ((inode_cor != inode_pai) || (dev_cor != dev_pai));

    p = nome_abs;
    *p++ = '/';

    for (; n > 0; ) {
        d = partes[--n];
        while (*d) *p++ = *d++;
        *p++ = '/';
    }
    strcpy(p, nome_rel);
    return(CORRETO);
}

int le_wmid(n, area)
int n;
long area;
{ int fm, nb;
  int i;
  fm = open(WORM_MID, LE);
  read(fm, &wmid, sizeof(wmid));
  if (wmid.nserie < n) {
      close(fm);
      return(ERRO);
  }
  do {
      if ((nb = read(fm, &wmid, sizeof(wmid))) == sizeof(wmid))
          if ( ((n == 0) || (n == wmid.nserie)) &&
              ((wmid.bloc_dir - wmid.ublus)*BLOCO_WORM >= area) ) {
              close(fm);
              return(wmid.dev);
          }
      } while (nb > 0);
  } while (nb > 0);
  return(-n);
}

char *dev_nome(dev)
int dev;
{ int i;
  for (i=0; dev_tab[i].dev; i++)
      if (dev_tab[i].dev == dev) break;
  return(dev_tab[i].nome);
}

```

wuser

```

/* wuser - Cadastrar Usuarios com Permissao de Acesso ao WORM */
/*
 * wuser.c 1.1 22/fev/93
 *
 * Luiz A. F. Bezerra
 *
 * Sintaxe:
 *
 * wuser <user_id> [area_Mbytes]
 *
 * Obs. 1. Somente com privilegios de super usuario, pode-se ativar
 * este procedimento
 */

#include <stdio.h>
#include <pwd.h>
#include "worm.h"

#define AREA_WORM 10 /* area padrao disponivel ao usuario */
#define CORRETO 1
#define ERRO 0
#define SUPER_USER 0 /* id atribuida pelo MINIX */

extern struct passwd *getpwnam();

int getareaworm();

reg_user user;

main(n_args, l_args)
int n_args;
char *l_args[];
{
    struct passwd *pwd;
    char nome[30]; /* nome do usuario -> logon */
    int fd;
    /*
     * Verifica se quem esta ativando o programa tem privilegio do
     * SUPER_USER
     */
    if (getuid() != SUPER_USER)
    {
        fputs("Acesso nao permitido \n", stderr);
        exit(ERRO);
    }
    /*
     * Verifica se foi especificada a identificacao do usuario
     */
    if (n_args < 2)
    {
        fputs("Faltou a identificacao do usuario \n", stderr);
        exit(ERRO);
    }
    /*
     * Pega o nome do usuario
     */
    strcpy(nome, l_args[1]);
    /*
     * Verifica se o usuario esta cadastrado
     */
    if ((pwd = getpwnam(nome)) == 0)
    {
        fputs("Usuario nao cadastrado \n", stderr);
        exit(ERRO);
    }
    user.uid = pwd->pw_uid;
    /*
     * Verifica se foi indicada a area disponivel no WORM para o usuario
     */
    if (n_args > 2)

```

```

        sscanf(l_args[2], "%d", &user.n_mb);
    else
        user.n_mb = AREA_WORM;
    /*
     * Verifica se existe area suficiente para este usuario
     */
    if (user.n_mb > getareaworm())
    {
        fputs("Area do WORM insuficiente \n", stderr);
        exit(ERRO);
    }
    /*
     * Acrescenta este usuario no arquivo wusers se ja nao existir
     */
    acresceus();

    exit(CORRETO);
}

getareaworm()
/*
 * Este procedimento desse consultar o sistema de arquivos WORM e
 * retornar a quantidade de area (Mbytes) disponivel no disco
 */
{
    return(10);
}

acresceus()
/*
 * Verifica se este usuario ja esta cadastrado.
 * Se estiver, altera se necessario a area em disco WORM
 * Se nao estiver, acrescenta o usuario a lista.
 */
/*
 * Estrutura do arquivo wusers
 */
/*
 * struct {
 *     int uid;
 *     n_mb;
 *     } *reg[];
 */
/*
 * Sendo que *reg[1] e usado para informar o numero atual de usuarios
 * existentes na lista.
 */
{
    int fd, i, n_users;
    long ret_reg;
    reg_user reg;

    fd = -1;
    ret_reg = -NB_REG_U;

    while (fd < 0)
    {
        /*
         * Abre o arquivo de permissao de acesso ao WORM
         */
        fd = open("/usr/worm/wusers", 2);
        /*
         * Se nao existe o arquivo, entao cria e inicializa numero de usuarios.
         */
        if (fd < 0)
        {
            fd = creat("/usr/worm/wusers", 0644);
            if (fd > 0)
            {
                reg.uid = 0; /* Numero de usuarios com permissao de acesso */
                reg.n_mb = 0;
                write(fd, &reg, NB_REG_U);
                close(fd);
                fd = -1;
            }
        }
        else
            return(ERRO);
    }
}
/*
 * Pega o numero atual de usuarios com permissao

```

```

*/
read(fd,&reg,NB_REG_U);
n_users = reg.uid;
/*
Verifica se este usuario ja existe na lista. Se existir atualiza a area.
*/
for (i=1;i<=n_users;i++)
{
    read(fd,&reg,NB_REG_U);
    printf("i: %d uid: %d mb: %d \n",i,reg.uid,reg.n_mb);
    if (reg.uid == user.uid)
    { reg.n_mb = user.n_mb;
      lseek(fd,ret_reg,1);
      write(fd,&reg,NB_REG_U);
      close(fd);
      return(CORRETO);
    }
}
/*
Acrescenta este usuario a lista
*/
write(fd,&user,NB_REG_U);
/*
Atualiza o numero de usuarios
*/
lseek(fd,0l,0);
read(fd,&reg,NB_REG_U);
n_users = ++reg.uid;
lseek(fd,ret_reg,1);
reg.uid = n_users;
write(fd,&reg,NB_REG_U);
lseek(fd,ret_reg,1);
read(fd,&reg,NB_REG_U);
close(fd);
printf("n_users: %d uid: %d n_mb: %d \n",reg.uid,user.uid,user.n_mb);
return(CORRETO);
}

```

APÊNDICE C

Procedimento para instalação do saa-WORM em disco magnético.

```
cd /usr          # Selecionar diretório /usr
mkdir worm       # Criar sub-diretório do WORM
cd worm          # Selecionar sub-diretório worm
#
mkdir crítico    # Criar lista de arquivos críticos
mkdir cópia      # Criar lista de arquivos normais
mkdir desuso     # Lista de arquivos não acessados 90 dias
wuser root       # Habilita super-usuário
```

Alteração necessária no arquivo /etc/rc do MINIX para ativação do saa-WORM:

```
/usr/worm/wgt # Gerenciador de transferência do WORM
/usr/worm/wdes # Coletor de arquivos normais -
               # não acessados a mais de 90 dias
```

Terminada a instalação, re-carregar o sistema MINIX.

COMANDOS DISPONÍVEIS AO USUÁRIO (HABILITADO)

Cadastrar (habilitar) um usuário para uso do saa-WORM.

wuser <identificação do usuário> [área]

A identificação do usuário deve ser exatamente igual a adotada no sistema MINIX.

A área (em megabytes) é um parâmetro opcional e possibilita ao super-usuário definir o uso do disco worm de acordo com a necessidade do usuário.

Copiar qualquer arquivo do disco magnético para o disco WORM (disquete).

wcp <identificação do arquivo>

A identificação do arquivo é a mesma usada no sistema de arquivos do MINIX.

Definir um arquivo crítico.

wci <identificação do arquivo crítico>

A identificação do arquivo é a mesma usada no sistema de arquivos do MINIX.

Desmontar (retirar de uso) uma mídia (disquete) do saa-WORM.

wdm <dispositivo MINIX>

O dispositivo MINIX pode ser qualquer um dos dispositivos utilizados para sistemas de arquivos em disquetes: /dev/fd0, /dev/fd1, /dev/at0 ou /dev/at1

Iniciar uma mídia (disquete) para uso no saa-WORM.

**wim <dispositivo MINIX> <tamanho em blocos>
<identificação para a mídia>**

O dispositivo MINIX pode ser qualquer um dos dispositivos utilizados para sistemas de arquivos em disquetes: /dev/fd0, /dev/fd1, /dev/at0 ou /dev/at1.

O tamanho em blocos indica o tamanho do disquete a ser utilizado: 360Kb ou 1,2 Mb.

A identificação para a mídia é de livre escolha do super-usuário.

Listar o diretório do WORM contendo todos os arquivos gravados nos disco WORM (disquetes).

wld

Listar somente o diretório correspondente a mídia ativa (disquete).

wlw

Montar nova mídia (disquete) no saa-WORM para uso.

wmm <dispositivo MINIX>

O dispositivo MINIX pode ser qualquer um dos dispositivos utilizados para sistemas de arquivos em disquetes: /dev/fd0, /dev/fd1, /dev/at0 ou /dev/at1

Re-instalar uma cópia de arquivo em WORM (disquete) para o disco magnético.

wrc <identificação do arquivo>

A identificação do arquivo é a mesma usada no sistema de arquivos do MINIX.