Universidade Federal de Pernambuco
Centro de Informática

Pós-graduação em Ciência da Computação

WFS: UM SISTEMA DE ARQUIVOS BASEADO NA POLÍTICA WRITE-ONCE READ-MANY NO ESPAÇO DO USUÁRIO

Tiago Lins Falcão

DISSERTAÇÃO DE MESTRADO

Recife
Março de 2011

Universidade Federal de Pernambuco
Centro de Informática

Tiago Lins Falcão

WFS: UM SISTEMA DE ARQUIVOS BASEADO NA POLÍTICA WRITE-ONCE READ-MANY NO ESPAÇO DO USUÁRIO

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: *Prof. Dr. Paulo Romero Martins Maciel*

Recife
Março de 2011

Catalogação na fonte
Bibliotecária Jane Souto Maior, CRB4-571

Falcão, Tiago Lins
WFS: um sistema de arquivos baseado na política Write-Once Read-Many no espaço do usuário / Tiago Lins Falcão - Recife: O Autor, 2011.
xx, 87 p. : il., fig., tab.

Orientador: Paulo Romero Martins Maciel.
Dissertação (mestrado) Universidade Federal de Pernambuco. CIn. Ciência da computação, 2011.

Inclui bibliografia e apêndice.

1. Avaliação de Desempenho. 2. Engenharia da computação. 3. Sistema operacional Linux. 4. Write-Once Read-Many. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004.029 CDD (22. ed.) MEI2011 – 030

Dissertação de Mestrado apresentada por **Tiago Lins Falcão** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**WFS: Um Sistema de Arquivos baseado na Política Write-Once Read-Many no Espaço do Usuário**", orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Djamel Fawzi Hadj Sadok
Centro de Informática / UFPE

Prof. Rivalino Matias Junior
Faculdade de Computação / UFU

Prof. Paulo Romero Martins Maciel
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 1 de março de 2011.

Prof. Nelson Souto Rosa
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Aos meus familiares e amigos.

AGRADECIMENTOS

Agradeço à minha mãe (Eloisa), aos meus avós (Eudes e Carmem), aos meus tios (Eudinho e Beta) e aos meus padrinhos (Cleones e Clarice), pela educação fornecida ao longo desses 25 anos. Saudades, meu padrinho!

Destino um agradecimento especial também às minhas irmãs: Gabriela, pelas horas passadas melhorando o português desse trabalho, sem a qual não poderia viver e à qual devo toda a minha vida (ela também está corrigindo isso!) e Carol, por me aturar, mesmo reclamando, nos momentos de estresse.

Após dois anos somando várias noites mal ou não dormidas, finais de semana dedicados às pesquisas, gostaria de agradecer à minha família por ser suporte em tempo integral.

A Helen, pela paciência, companheirismo, cumplicidade e pelos tantos momentos de felicidade durante os nossos três primeiros anos juntos.

Aos meus amigos, pela paciência e apoio que me dedicaram. Dentre eles: Ermeson Carneiro, Rubens Matos, Jair Figueirêdo, Manuela Braga, Paula/Laura Freitas, Gabriela Braga, Juliana Sauvé, Eduardo Cabral, Erica Sousa, Eduardo Tavares, Gustavo Callou, Julian Menezes, Gracieth Mendes, Bruno Nogueira, Kalil Bispo e Thiago Viana.

A FACEPE pelo auxílio financeiro ao longo desses 2 anos.

Ao meu orientador Prof. Paulo Maciel, pelo apoio, disponibilidade, comentários imprescindíveis, dedicação ao desenvolvimento desta pesquisa e por ter acreditado acima de tudo em mim e no meu trabalho.

RESUMO

O constante crescimento da quantidade de dados armazenados em computadores trouxe a necessidade de assegurar a integridade e a confiabilidade dessas informações. Dessa forma, princípios como os do *Write-Once Read-Many* (WORM) podem ser aplicados para reduzir o risco associado à perda de informações involuntariamente.

Este trabalho apresenta um sistema de arquivos WORM, o WFS, implementado a partir da infraestrutura FUSE. Este sistema fornece os recursos fundamentais WORM, por exemplo, não permitindo que os arquivos e diretórios sejam renomeados ou removidos ou que haja a utilização de comandos de superusuário. O WFS possui o foco em usuários domésticos, que precisam armazenar dados como e-mails, fotos, músicas, sem um alto investimento financeiro.

Como principais requisitos do sistema desenvolvido, podemos citar: ser código-livre; atender a usuários sem privilégios de administração; ser instalado e configurado facilmente; apresentar desempenho satisfatório; possibilitar uma forma alternativa de modificação e de remoção; permitir o gerenciamento dos dados de forma simples.

Por fim, uma avaliação de desempenho foi realizada com o objetivo de reduzir a perda de desempenho em relação ao Ext3. Para tal, técnicas de experimento fatorial 2^k foram utilizadas. Ficou constatado que, para a carga de teste utilizada, o WFS introduziu uma perda de desempenho inferior a 12% quando comparado ao Ext3 não-WORM.

Palavras-chave: Avaliação de Desempenho; Write-Once Read-Many; FUSE; Linux.

ABSTRACT

The constant growing of the data quantity kept in computers brought the necessity of assuring the integrity and reliability of these pieces of information. Thus, principles like the ones from Write-Once Read-Many (WORM) can be applied to reduce the risk associated to the involuntary loss of information.

This work proposes a WORM file system, the WFS, implemented from the FUSE infrastructure. This system provides the fundamental WORM resources, for example, not allowing that the files and directories are renamed or removed or that superuser commands are executed. The WFS focus on domestic users, that need to keep data like e-mails, pictures, songs, without a high financial investment.

As main requisites from the developed system, we can cite: be free-code; serve users without administrations privileges; be easily installed and set up; present satisfactory performance; allow an alternative way of changing and removing; allow the growing of data in a simple way.

Ultimately, a performance assessment took place with the aim of reducing the loss of performance in relation to Ext3. For such, techniques of factorial experiment 2^k were used. It was verified that, considering the used workload, the WFS generated a loss of performance under 12% comparing to Ext3 not-WORM.

Keywords: Performance Evaluation; Write-Once Read-Many; FUSE; Linux.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ADS - *Avaliação de Desempenho de Sistemas.*

API - *Application Programming Interface.*

EncFS - *Encrypted Filesystem.*

Ext3 - *Third Extended File System.*

FS - *File System.*

FUSE - *Filesystem in Userspace.*

GCC - *GNU Compiler Collection.*

HD - *Hard Disk.*

HP - *Hewlett-Packard.*

I/O - *Input/Output.*

MEC - *Ministério da Educação.*

POSIX - *Portable Operating System Interface.*

RAM - *Random Access Memory.*

ROFS - *Read-Only File System.*

SCSI - *Small Computer System Interface.*

SO - *Sistema Operacional.*

UDO - *Ultra Density Optical.*

VFS - *Virtual File System.*

WFS - *Write-Once Read-Many File System.*

WORM - *Write-Once Read-Many.*

CAPÍTULO 1

INTRODUÇÃO

Este capítulo provê uma breve introdução aos sistemas *Write-Once Read-Many*, contextualizando as principais necessidades do público-alvo desse trabalho. Em seguida, são apresentados a motivação, os trabalhos relacionados e a proposta deste trabalho bem como o seu escopo.

1.1 CONTEXTO

Segundo dados do Ibope/Nielsen [IBO10], em 2009 aproximadamente 36% dos domicílios brasileiros possuíam computadores, além disso, o país apresentava, em dezembro de 2009, cerca de 67,5 milhões de internautas acima de 16 anos. Para 2014, o governo brasileiro tem como meta levar banda larga de pelo menos 1 Mbps a todos municípios brasileiros a preços acessíveis, conectando cerca de 50% das residências à rede mundial de computadores [Pav10a]. Embora seja um processo mais lento se comparado com a inclusão digital alcançada por países mais avançados tecnologicamente, o esforço de propiciar às pessoas acesso aos meios digitais pode ser comprovado por projetos como: *Cidadão Conectado - Computador para Todos* [Fed10a], responsável por baratear os custos dos computadores e facilitar as condições de pagamento; *Programa Computador Portátil para Professores* [Fed10b], destinado a "facilitar a aquisição de computadores portáteis para professores da rede pública e privada da educação básica, profissional e superior, credenciadas junto ao MEC, a baixo custo e condições diferenciadas de empréstimo".

Juntamente com os computadores, o governo, visando reduzir os gastos com licenças de *software* proprietários, procura disseminar também a utilização de *softwares* livres entre a população, importante incentivo para a expansão da utilização dos sistemas operacionais baseados em Linux. Essa política possibilita, aos usuários de computador, realizar de forma simples e eficiente diversas ações do cotidiano, como, por exemplo, elaboração de documentos, gerenciamento das despesas da casa, pagamento de contas, comunicação interpessoal etc. Tais atividades, quando executadas a partir de computadores, geram aos usuários residenciais a necessidade de manter dados armazenados nesses dispositivos. Essas informações guardadas podem variar desde simples arquivos de música a exames médicos, extratos bancários, dados de imposto de renda etc.

Contudo, as estratégias tradicionais utilizadas pelos sistemas operacionais para o armazenamento de dados em discos rígidos (HD) podem trazer alguns inconvenientes aos usuários, como, por exemplo, o risco de remoções involuntárias e de perda de dados por danos em setores do HD. Para minimizar riscos de perda de dados, os usuários residenciais têm como opção utilizar dispositivos de armazenamento como CD-R e DVD-R para armazenar seus dados importantes. No entanto, essas mídias possuem inconvenientes

que muitas vezes desestimulam certos usuários a utilizá-las na realização das cópias de segurança. O baixo desempenho e a restrição de espaço em relação ao disco rígido, assim como a necessidade de um *software* específico para gravação de conteúdo[1] são alguns dos pontos negativos da utilização das mídias ópticas. Outro fator que pode contribuir para essa rejeição é o fato desses discos serem dispositivos externos ao computador, necessita de aquisição e de armazenamento em local diferenciado.

No contexto das corporações, a gestão da informação tornou-se uma área importante e regulamentada na maioria dos países [OFK04] [Par09], visto que, em muitos casos, empresas necessitam preservar, por um longo tempo, dados financeiros, imagens médicas, e-mails, documentos de certificação etc. Assim, para se adequarem as regulamentações, técnicas de armazenamento de informações precisaram ser aprimoradas. Hasan et al. [HTS+05] destacam que um dos principais modos de atender aos requisitos de confiabilidade e de durabilidade é adicionar ao conteúdo armazenado a propriedade da imutabilidade, ou seja, não deve ser possível modificar a informação guardada em um período posterior a inclusão dessa propriedade.

No âmbito dos sistemas de arquivos, a imutabilidade impede a ocorrência de qualquer alteração no conteúdo dos arquivos. A política WORM (*Write-Once Read-Many*) pode ser considerada uma aplicação prática da propriedade da imutabilidade, pois viabiliza o armazenamento de informações de forma *on-line* [2], impedindo a remoção ou a modificação de dados. Dispositivos WORM têm sido amplamente utilizados por corporações para o arquivamento de dados que exijam um longo período de vida, visto que, em muitos casos, devem estar aptos a serem auditáveis [WZ03]. Portanto, é de se esperar a existência de uma rigorosa política que gerencie o acesso aos dados e que proíba a destruição e a alteração do conteúdo armazenado [SW07].

Assim, pela etimologia do termo em inglês, dá-se o nome de dispositivo WORM àquele que permite a escrita de dados em disco e impede que os mesmos sejam modificados ou apagados. Tal técnica não é completamente desconhecida dos usuários residenciais, visto que, muitos deles já mantêm o hábito de gravar seus dados em CD-R ou DVD-R. No entanto, é importante ressaltar que os usuários optam pela utilização de discos rígidos como principais dispositivos de retenção de dados, por serem uma forma mais intuitiva, rápida e conveniente do que as mídias ópticas para armazenamento de informações, além de propiciar maior espaço de armazenamento [Sil10, WZ03].

Ao perceber um nicho de mercado em aberto, a *EMC Corporation* [Cor10c] firmou um projeto de cooperação com a UFPE, cujo principal foco foi desenvolver um sistema de arquivos com características WORM, de código-livre, voltado aos anseios dos usuários residenciais Linux[3]. Com o objetivo de facilitar o administração dos arquivos por parte de usuários residenciais, possivelmente sem prioridade de administrador, o presente trabalho

[1] O processo de escrita de dados em dispositivos como CD-R e DVD-R requer do usuário domínio de aplicações singulares como o Nero [Ltd10], PowerISO [PC10] etc.

[2] O termo *on-line* refere-se ao armazenamento em tempo de execução.

[3] Embora a solução proposta apresente portabilidade entre diversos SOs, o foco inicial do projeto foi oferecer soluções para usuários Linux.

apresenta um sistema de arquivos virtual com características WORM, chamado WFS.

1.2 MOTIVAÇÃO

Embora a utilização de discos rígidos ofereça diversas vantagens em relação às mídias ópticas, não é raro àqueles que se utilizam da tecnologia magnética passar pela experiência de perder informações importantes, seja por erros no armazenamento em *hardware*, seja por remoções ou por sobrescritas equivocadas de arquivos. Os sistemas operacionais atuais procuram reduzir o risco de remoção e de sobrescrita involuntárias utilizando-se de telas com o objetivo de confirmar se o usuário, de fato, deseja realizar tal procedimento. No entanto, em muitos casos, essa proteção não é suficiente e os arquivos são perdidos de modo definitivo. Os usuários mais avançados podem optar por gerenciar manualmente as permissões dos arquivos, no entanto, essa escolha exige muita atenção e frequente trabalho adicional. O gerenciamento dos arquivos pode ser ainda mais difícil, caso o usuário não possua permissão de administrador do sistema.

Assim, este trabalho tem seu foco voltado a um nicho pouco explorado por fabricantes de soluções em WORM, cujos requisitos foram elicitados a partir das reuniões semanais com integrantes da empresa. O público-alvo do WFS são usuários finais Linux que desejem manter os seus dados em disco rígido, reduzindo a chance de remoções involuntárias dos arquivos, sem que, para isso, tenham de investir em *hardware* específico, envolvendo, portanto, investimento financeiro adicional. Dessa forma, sugere-se um sistema de arquivos virtual de código livre com características WORM, desenvolvido a partir da infraestrutura FUSE (*FileSystem in Userspace*) [Sze09]. O WFS bloqueia operações de administrador (sudo, chmod, chown etc.) [Inc10], permite a escrita de conteúdo por parte do usuário, mas impede que os dados escritos sejam renomeados, modificados ou removidos acidentalmente através de acesso convencional ao sistema de arquivos.

Os principais requisitos do sistema desenvolvido são: ser código-livre; atender a usuários sem privilégios de administração; ser instalado e configurado em poucos passos; apresentar desempenho similar a um sistema de arquivos nativo (não-WORM); possibilitar uma forma alternativa de modificação e de remoção de dados; além de permitir o gerenciamento dos dados de forma simples.

A utilização de sistemas de arquivos de espaço do usuário, tais como os implementados a partir do FUSE, propicia um conjunto de benefícios aos desenvolvedores e aos usuários, como a facilidade de desenvolvimento (abstração de operações de baixo nível), a portabilidade entre sistemas operacionais e a possibilidade de utilização por parte de usuários sem privilégios de administrador. Para muitas aplicações, essas vantagens superam o encargo gerado pela pequena perda de desempenho dos sistemas em *user-level* se comparado aos sistemas de arquivos nativos do sistema operacional.

1.3 TRABALHOS RELACIONADOS

Existem trabalhos na literatura que reportam a implementação de sistemas de arquivos com características WORM [FKM00], [QD02], [SWZ04], [SWZ05a], [Sio08], [MW06], [HWS07], [MWHM07], [SW07]. No entanto, nenhum deles relata um sistema de arquivos de código-aberto, em que os usuários Linux, possivelmente sem permissão de administrador, possam se beneficiar de características WORM. É importante ressaltar que o sistema de arquivos desenvolvido neste trabalho necessita de que os usuários possuam apenas o *FUSE* (*Filesystem in Userspace*) instalado. Maiores detalhes sobre o FUSE serão apresentados no Capítulo 2.

Considerando soluções em espaço do usuário, dois *filesystem* merecem destaque devido a algumas características desejáveis ao WFS: o *Read-Only Filesystem for FUSE* [Kel10] e o ChironFS [Fur10]. O primeiro é um sistema de arquivos no espaço do usuário que utiliza a infraestrutura FUSE [Sze09] para fornecer aos usuários características que minimizam o risco de remoções involuntárias de arquivos, através de um sistema de arquivos focado apenas em leitura de conteúdo. Embora o *ROFS* satisfaça algumas características desejadas ao sistema, como apresentar desempenho que atende aos requisitos definidos pelo cliente (*EMC Corporation*), evitar remoções involuntárias dos arquivos, esse sistema não propicia as características WORM definidas pelo cliente[4], pois a criação de conteúdo não é permitida, os usuários podem apenas ler o que foi previamente armazenado na partição. Já o ChironFS, trata-se de um sistema de arquivos baseado no FUSE cujo o foco é propiciar garantias de disponibilidade aos usuários através de replicação por *software* utilizando técnicas baseadas em RAID 1. Embora o risco de perda de informações devido à corrupção de dados ou à falha de *hardware* seja reduzido, é importante destacar que a tecnologia RAID 1 necessita que 50% do espaço total de armazenamento seja reservado para gravação da réplica do conteúdo, aumentando consideravelmente o custo para armazenagem de informação, bem como o tempo para a criação da cópia nas operações de escrita. Além dessas desvantagens, o ChironFS não possui nenhuma característica WORM.

Sion [Sio08] apresenta um sistema de armazenamento WORM que proporciona garantias de retenção e de migração dos dados, focando em soluções por *hardware*. Este foco de abordagem tem como objetivo principal impedir modificação de conteúdo. Além disso, a solução impede que os dados sejam removidos antes do ciclo de vida pré-estabelecido. Apesar de oferecer garantias fortes de retenção dos dados, esta solução envolve uma arquitetura de *hardware* de difícil implementação ou mesmo aquisição no mercado. Logo, os requisitos do sistema referentes ao baixo custo de aquisição e à simplicidade para instalação e configuração não são atingidos por esta solução.

Quinlan et al. [QD02] relatam um sistema de armazenamento em rede, denominado Venti. Esse sistema proporciona um repositório de arquivos cuja remoção de dados é proibida. Dados podem ser compartilhados por várias aplicações-cliente. Essa abordagem impõe uma política de gravação única que impede a destruição acidental ou maliciosa de

[4]Os requisitos do sistema de arquivos desenvolvido neste trabalho pode ser encontrado na Seção 3.1.1

dados. Em suma, Venti é um sistema em rede destinado ao arquivamento de conteúdo, identificando os blocos dos dados por um *hash* do seu conteúdo. O principal benefício dessa abordagem é a verificação da integridade de dados, porque quando um bloco é recuperado, tanto o cliente quanto o servidor podem computar o *hash* dos dados e compará-lo com o *hash* solicitado. Essa operação permite que o cliente evite erros provenientes da não-detecção de dados corrompidos e habilita o servidor a identificar quando a recuperação do erro é necessária. Embora a abordagem desenvolva os principais conceitos de soluções WORM, não é adequada para os usuários finais Linux, pois é uma solução para o sistema operacional Plan9 [Bel09].

O projeto e a implementação de um sistema de arquivos distribuído que garante a um grande número de clientes o acesso somente de leitura aos dados é descrito em Fu e Kaashoek [FKM00]. Nessa abordagem, o administrador cria um banco de dados seguro para o conteúdo do sistema de arquivos (usando técnicas de criptografia). O administrador, então, replica o banco de dados em sistemas não-confiáveis [5]. Nesses sistemas, um servidor (*software*) fornece o conteúdo da base de dados aos clientes sem acesso à chave privada do sistema de arquivos. Um programa cliente verifica a autenticidade dos dados antes de responder ao usuário. A abordagem tem alguns inconvenientes quando são consideradas as atualizações dos arquivos do sistema, devido ao fato de que quando uma alteração nos dados acontece, o administrador tem de gerar um novo banco de dados e enviá-lo para as réplicas do servidor. Sendo assim, o sistema não implementa de forma eficiente a característica de escrita necessária para WORM. Além disso, as operações de escrita só podem ser efetuadas por usuários com privilégios de administrador.

Yliluoma [Yli09] apresenta um sistema de arquivos somente-leitura para o Linux, o Cromfs, cujo o foco principal é a compressão dos dados armazenados. Cromfs usa o LZMA (*Lempel-Ziv-Markov chain-Algorithm*) do 7-zip [Pav10b], e um mecanismo de *merging* dos blocos. Nesse sistema de arquivos, os dados são divididos em fragmentos e esses, por sua vez, são armazenados como deslocamentos (*offsets*) de blocos (chamados de *fblocks*). Em outras palavras, as partes de arquivos diferentes são compactadas em um mesmo trecho para alcançar uma compressão eficaz. Além disso, fragmentos idênticos são compactados apenas uma vez. Segundo o relato, arquivos e até mesmo partes de arquivos duplicados são detectados e armazenados apenas uma vez. Esse sistema de arquivos tem dois inconvenientes principais: (i) ser mais lento do que outras soluções como SquashFS [Lou08] e CramFS [Qui02], e (ii) usar muita memória, devido à necessidade constante de compactação de dados. Embora este trabalho seja um sistema de arquivos de código-aberto Linux que impede a modificação de conteúdo, o mesmo não permite ao usuário adicionar informações *on-line* e possui uma queda de desempenho causada pelas operações de compactação de conteúdo.

Uma implementação de um sistema de arquivos com características WORM com foco em segurança e independente do sistema operacional é descrita por Wang e Zheng [WZ03]. O foco principal dessa abordagem é combater invasores com privilégios de administração do sistema, visto que eles podem, inclusive, excluir os arquivos de *log* contendo

[5] Sistemas não-confiáveis são aquelas que não oferecem garantias de seus serviços.

informações sobre os ataques. Com o objetivo de prevenir tanto de ataques internos quanto de externos, Zheng sugere uma arquitetura *append-only* que se utiliza de discos magnéticos. Essa arquitetura fornece garantias WORM através de modificações dos *device drivers* dos dispositivos de armazenamento. Embora seja uma abordagem interessante do ponto de vista acadêmico, tal solução não supre alguns requisitos do sistema (ver Seção 3.1.1): (i) por ser uma solução a ser implementada no núcleo do SO, dificulta a instalação e a utilização por usuários sem privilégios de administração; (ii) não foi encontrada uma versão compatível com código padrão do Linux; (iii) não possibilita uma forma alternativa de remoção e modificação do conteúdo armazenado.

O Ext3 propicia um mecanismo para tornar arquivos imutáveis, os atributos *chattr* e *lsattr*. Contudo, esta solução não está de acordo com os requisitos do sistema especificados pelo cliente, visto que a utilização desses atributos torna o gerenciamento de arquivos uma operação ainda mais complexa para os usuários domésticos, além de não propiciar forma alternativa para remover/modificar um arquivo que foi criado de forma equivocada [HTS+05].

Russo [GR+95] descreve um sistema de arquivamento, de propósito geral, seguindo a política *Write-Once Read-Many* independente de sistema operacional. Esta solução utiliza como mídia de arquivamento os discos ópticos, aproveitando da característica WORM natural desses componentes. Contudo, a utilização desses discos vai de encontro a alguns requisitos estabelecidos pelo cliente, visto que: (i) impõe uma modificação de *hardware* ao usuário; (ii) aumenta a complexidade na manutenção do sistema e (iii) gera um aumento no custo para o arquivamento das informações, visto que, normalmente, mídias ópticas apresentam preços mais elevados que a solução magnética.

1.4 OBJETIVOS

Este trabalho propõe o WFS, um sistema de arquivos no espaço do usuário voltado àqueles que desejam reduzir o risco de perda de dados ocasionado por alterações ou remoções involuntárias de arquivos. O trabalho apresenta os requisitos do sistema, elicitados a partir das reuniões semanais do projeto de colaboração entre o Centro de Informática da Universidade Federal de Pernambuco e a *EMC Corporation*, bem como detalhes de implementação do WFS.

Para garantir que o sistema atenda aos requisitos, uma avaliação de desempenho foi realizada. Essa avaliação efetuou comparações entre o WFS e o Ext3, um sistema de arquivos nativo do Linux sem suporte às características WORM. Os resultados dessa fase também são discutidos neste trabalho.

Mais especificamente, este trabalho tem os seguintes objetivos:

- reunir fontes públicas de pesquisa para encontrar e avaliar diferentes soluções baseadas em tecnologia WORM, considerando particularmente ambientes Linux;
- fornecer recomendações para se adotar ou projetar implementações WORM;

- desenvolver a primeira versão de código-aberto do WFS, com o foco em usuários residenciais;
- caracterizar o desempenho desse Sistema de Arquivos WORM quando comparado a um sistema de arquivos padrão R/W Linux;
- definir recomendações para ajustes de configuração no WFS, objetivando melhorar o desempenho de I/O.

1.5 ESTRUTURA DA DISSERTAÇÃO

Além da seção introdutória, este trabalho está organizado da seguinte forma: no Capítulo 2, é apresentado um breve referencial teórico, com os conceitos de *Write-Once Read-Many*, sistemas de arquivos nativos e em espaço do usuário, FUSE e avaliação de desempenho de sistemas, que são referências básicas para a realização e a compreensão deste estudo; no Capítulo 3, são apresentados os principais requisitos do sistema desenvolvido, o WFS, assim como detalhes de implementação do mesmo; o Capítulo 4 descreve os resultados da avaliação de desempenho do sistema de arquivos WORM; por fim, o Capítulo 5 apresenta as conclusões e os trabalhos futuros da referida dissertação de mestrado.

CAPÍTULO 2

FUNDAMENTOS

Este capítulo apresenta os principais conceitos sobre dispositivos *Write-Once Read-Many*, assim como as características, a classificação e os requisitos dessas soluções. Em seguida, são abordados os sistemas de arquivos, bem como o FUSE. Finalmente, as principais fontes sobre Avaliação de Desempenho de Sistemas relacionadas a esse trabalho são referenciadas.

2.1 WRITE-ONCE READ-MANY

Atualmente, as informações relacionadas a uma variedade de temas como dados pessoais, assuntos políticos, conteúdos financeiros e médicos, etc. são consideradas confidenciais e, consequentemente, um número limitado de pessoas estão autorizadas a acessá-las. Esses dados poderão ainda ter restrições mais rigorosas, como, por exemplo, a impossibilidade de serem alterados ou eliminados antes do final de um determinado período de vida [SW07].

A área de gestão da informação assumiu grande importância e atualmente é regulamentada na maioria dos países. O *Health Insurance Portability and Accountability Act* (HIPAA) foi promulgado pelo Congresso americano em 1996 [OFK04], e é utilizado para reduzir os custos e os encargos administrativos nos cuidados de saúde, melhorando a eficiência e eficácia do sistema de saúde por meio da padronização do intercâmbio eletrônico de dados para determinadas operações administrativas e financeiras. Além disso, ele ajuda a garantir a proteção da privacidade dos registros pessoais de saúde dos americanos, fornecendo a segurança e a confidencialidade necessárias para essas informações [OFK04]. A União Européia, por sua vez, estabeleceu diretivas relativas à proteção de dados pessoais, oferecendo garantias de segurança e de privacidade a informações pessoais [Par09].

Um modo constantemente utilizado para atender às normas de proteção de dados é a aplicação da propriedade de imutabilidade, ou seja, utilizar de mecanismos que impedem a realização de quaisquer modificações subsequentes na informação a ser protegida. Em relação aos sistemas de arquivos, a imutabilidade está associada à prevenção de qualquer alteração no conteúdo dos arquivos. A legislação que regula o *copyright* [Org10] e a propriedade intelectual [dRFdB98] requer a aplicação dos conceitos de imutabilidade, pois torna necessário verificar e garantir a autenticidade de muitos arquivos de multimídia. Sistemas de armazenamento que garantem a propriedade da imutabilidade dos arquivos também asseguram a integridade dos dados [WZ03, HTS+05].

A proliferação da Internet nas últimas décadas criou novas oportunidades de ar-

mazenamento e de distribuição de conteúdo. Ao mesmo tempo, essa expansão digital propiciou um aumento nos níveis de invasão e de ataques a sistemas computacionais [HTS+05, HSW07]. Esses "invasores", em muitos casos, conseguem obter privilégios de administração do sistema, condição suficiente para modificar ou remover arquivos que deveriam ser mantidos livres de tal risco.

Em prol da segurança do sistema, as operações realizadas sobre os dados devem ser registradas de modo confiável, permitindo consulta posterior. Muitas ferramentas de detecção de intrusão necessitam dos arquivos de log das atividades do sistema e do usuário para executarem suas atividades. Contudo, há o risco de que invasores consigam apagar (ou adulterar) esses logs após uma invasão bem-sucedida, isso inviabiliza a detecção do ataque. Portanto, é cada vez mais frequente a necessidade de desenvolver ferramentas de armazenamento que sejam capazes de evitar modificação no conteúdo armazenado no sistema, protegendo a informação. Especificamente, os benefícios da criação de um registro completo, inalterável e permanente de todas as atividades em um sistema de computador são, pelo menos, os seguintes [WZ03]:

- Sistemas de detecção de intrusão podem consultar os registros para detectar ataques;
- Realização de auditorias com mais segurança, visto que os dados são mais confiáveis quando possuem a propriedade da imutabilidade [Par09] [OFK04];
- Inibição de ataques. Como não é possível/viável remover os rastros da invasão que estão armazenados no log de operações, indivíduos mal-intencionados - com receio de serem detectados e identificados - podem hesitar em atacar o sistema.

Os dispositivos baseados na política WORM (*Write-Once Read-Many*) são uma frequente aplicação comercial da propriedade da imutabilidade. Os WORM *Storages* são dispositivos de armazenamento de dados que permitem a escrita de informações em disco e impedem que os dados sejam modificados ou apagados. Esses dispositivo têm sido historicamente utilizados para arquivamento de informações que requerem um longo período de vida [WZ03, GR+95]. Nos últimos anos, os usuários domésticos tiveram a possibilidade e optar pela utilização de dispositivos baseados na política WORM para manter seus arquivos - como documentos financeiros, e-mails, músicas, fotos, entre outros - com segurança. Os CD-R e DVD-R são exemplos de unidades de armazenamento que fornecem características WORM. No entanto, é importante ressaltar que a utilização de um disco rígido como dispositivo de armazenamento de dados é uma forma mais rápida, conveniente e intuitiva[1] para os usuários.

[1]Este trabalho considera o disco rígido como mídia de armazenamento mais intuitiva do que os discos ópticos, pois o usuário pode armazenar os dados diretamente na mídia através do gerenciador de arquivos padrão do sistema operacional (Nautilus ou Windows Explorer, por exemplo). Enquanto, a escrita no CD-R e DVD-R ocorre através de *softwares* diferenciados, como o Nero [Ltd10] ou PowerISO [PC10].

2.1.1 Classificação das Soluções WORM

Williams [Wil96], Wang [WZ03] e Hasan [HTS+05] classificam as soluções baseadas na política *Write-Once Read-Many* em três grandes categorias:

- Physical WORM ou P-WORM
- Embedded WORM ou E-WORM
- Software WORM ou S-WORM

As mídias ou dispositivos que possuam, por natureza, a propriedade da imutabilidade são classificados como soluções P-WORM. Nessas soluções, as características WORM são intrínsecas ao material a partir do qual são fabricados, fornecendo, portanto, proteção eficaz contra modificação de conteúdo em ataques maliciosos. Duas tecnologias diferentes são amplamente utilizadas para confecção dos P-WORM, os discos ópticos e os magneto-ópticos.

Atualmente o tipo mais comum de WORM físico são os discos ópticos, por exemplo CD-R e DVD-R. Esses discos garantem a propriedade da imutabilidade aos dados que armazenam e possuem tamanhos comerciais que variam entre 670MB até 8GB. Em 2007, a HP e a Plasmon desenvolveram um disco, UDO (*Ultra Density Optical*), capaz de armazenar mais de 120GB [WZ03]. Segundo a *Data Archive Corporation* [Cor10a], há previsão de que esta capacidade chegue aos 240GB no ano de 2012. Os discos ópticos com propriedade WORM apresentam tempo de acesso muito mais rápido que dispositivos de fita magnética, além de possuírem outro diferencial: possibilitar o acesso randômico e não apenas sequencial.

Mídias magneto-ópticas também são utilizadas em soluções P-WORM. Para tal, é necessária uma modificação, em temperaturas particulares, das propriedades magnéticas de determinadas mídias. O aquecimento da superfície dos discos é controlado opticamente usando *laser*. A mídia só pode ser escrita quando aquecida a uma temperatura especificada pelo fabricante. Estes discos magnéticos-ópticos toleram mais danos mecânicos que as mídias ópticas. Um ponto favorável para os usuários das mídias magneto-ópticas é a velocidade das operações de leitura, que acontecem à velocidade dos discos magnéticos, ou seja, consideravelmente mais rápidos que os discos ópticos convencionais. Atualmente, existem soluções comerciais que armazenam até 9GB [Max10].

A maior vantagem da tecnologia P-WORM é a ampla produção por diferentes fabricantes, haja vista a popularidade dos CD-R e DVD-R. Outro ponto positivo é o preço de comercialização desses dispositivos, que são bem mais baixos se comparados a outras soluções WORM. O maior problema de discos ópticos P-WORM é a velocidade da operação de escrita, bastante inferior se comparada à escrita em dispositivos magnéticos. Além do mais, a capacidade de armazenamento dos dispositivos ópticos comerciais é consideravelmente inferior a dos discos rígidos convencionais (não-WORM).

Nos dispositivos do tipo E-WORM, o *device driver* e o *firmware* do disco são implementados de forma a prover garantias WORM a mídias que não possuem intrinsecamente

características WORM. Logo, discos rígidos não-WORM podem ser utilizados desde que, por exemplo, as restrições de imutabilidade sejam aplicadas no *device driver* do dispositivo de armazenamento. Essa é uma solução bastante popular no meio corporativo, visto que diversos fabricantes oferecem soluções proprietárias baseadas em fitas magnéticas ou discos magnéticos.

As soluções WORM baseadas em dispositivos magnéticos de armazenamento normalmente aplicam as técnicas de proteção de dados através de um *firmware*, visando impedir a modificação de conteúdo. O EMC Centera [Cor10b] e o Network Appliance product NetStore/Snaplock [Lue03] merecem destaques como exemplos de produtos que se apresentam como soluções WORM embarcadas baseados em uma abordagem híbrida entre *firmware* (E-WORM) e *software* (S-WORM).

Dispositivos que se utilizam da tecnologia E-WORM possuem espaço de armazenamento consideravelmente superior aos discos ópticos, além de realizar operações de escrita mais rapidamente que as mídias P-WORM tradicionais. Considerando a capacidade, o custo comparativo também é menor [WZ03]. Contudo, as soluções por *software* possuem a desvantagem das mídias magnéticas: são mais suscetíveis a problemas devido a danos de setores físicos dos discos rígidos. As soluções WORM baseadas em fitas magnéticas, por sua vez, permitem apenas leituras/escritas sequenciais, o que dificulta o acesso à informação. Por fim, soluções que utilizam dispositivos de armazenamento sem características WORM intrínsecas estão suscetíveis a invasores que, obtendo acesso físico ao dispositivo de armazenamento, podem danificá-lo, destruindo o conteúdo.

Através da utilização de mecanismos como sistemas de arquivos modificados, o sistema operacional (SO) propicia proteção WORM conhecida por S-WORM. Logo, o presente trabalho propõe uma solução WORM desse tipo, um sistema de arquivos WORM no espaço do usuário (ver Seção 2.3) com o foco nas necessidades de usuários residenciais.

A maior vantagem de utilizar a tecnologia S-WORM é a facilidade de implementação e de integração com o sistema operacional. Também vale ser ressaltado que, por ser implementado em mais alto nível, é possível adequar o sistema desenvolvido aos seus requisitos de forma mais simples. No entanto, por serem soluções aplicadas apenas nos sistemas de arquivos a segurança contra invasões é bastante limitada, tornando praticamente inviável a utilização em ambientes corporativos com rígidas normas para preservação de conteúdo.

Wang [WZ03] classificou as principais soluções WORM de acordo com o tipo de tecnologia utilizada. A Tabela 2.1 traz um resumo da classificação realizada por esse autor, adicionando a ela algumas soluções WORM apresentadas neste trabalho, não descritas no trabalho original.

2.1.2 Características e Requisitos WORM

Esta seção discorre sobre os requisitos e as características WORM mais comumente encontrados na literatura [WZ03, Sio08, HWS07]. No entanto, dependendo do domínio da aplicação WORM, alguns desses requisitos podem não ser relevantes, logo, não terem

Tabela 2.1: Classificação das soluções WORM

Solução WORM	Tipo	Características
Disco magneto-óptico [Max10]	P-WORM	Armazenamento de até 9GB.
CD-R, DVD-R, UDO	P-WORM	Armazenamento até 120GB.
Advanced Packet Vault (APV) [ACF01]	P-WORM	Solução WORM capaz de detectar intrusão, mas possui lentidão e limitação de espaço devido ao uso de CD-R [WZ03].
RUSSO [GR+95]	P-WORM	Solução WORM de propósito geral baseada em discos ópticos, independente do sistema operacional.
Strong WORM [Sio08]	P-WORM e E-WORM	Proporciona garantias de retenção e de migração dos dados, focando em soluções por *hardware*.
SnapLock TM WORM storage system [Lue03]	E-WORM	Permite a seus usuários criar partições em servidores de arquivos NearStore com características WORM.
EMC Centera [Cor10b]	E-WORM	Sistema WORM baseado em discos rígidos. Foco em desempenho e em espaço de armazenamento. Principal ponto fraco é ser suscetível a ataques.
Wang [WZ03]	E-WORM	Sistema *append-only* com o foco em segurança, o que possibilita combater ataques, sejam eles internos ou externos.
Atributos *chattr* e *lsattr* [HTS+05]	S-WORM	Dificuldade de gerenciamento dos arquivos por leigos. Segurança limitada.
Linux Intrusion Detection System (LIDS) [LID10]	S-WORM	Permite que arquivos sejam criados em modo apenas leitura ou apenas acréscimo de conteúdo (*append-only*)

necessidade de implementação.

Integridade e Confiabilidade. Referem-se à confiança e à segurança do dado mantido no sistema, visto que, com o advento do armazenamento em rede e o aumento da complexidade dos *storages*, novos modos de falha estão sendo observados. Assim, dife-

rentes desafios surgem para garantir a integridade dos dados. Muitas são as causas que podem alterar o conteúdo dos dados armazenados. Os dados podem ser corrompidos devido a avarias de *hardware* ou *software*, como também através de ações maliciosas realizadas por agentes externos ou internos [SWZ05a]. Logo, o sistema de armazenamento WORM deve garantir a integridade dos dados com atenção especial a práticas de pessoas mal-intencionadas.

Retenção Controlada. Em dispositivos WORM, os dados podem ser gravados uma vez, mas as operações de leitura de conteúdo podem ser executadas ilimitadamente. No entanto, não podem ocorrer alterações ou remoções não detectadas nos dados antes do fim do ciclo de vida estabelecido. Os principais adversários dessa exigência são invasores com poder de administração do ambiente, visto que têm poderes de superusuário e acesso físico ao *hardware* do sistema de armazenamento.

Remoção Segura. Uma vez que um registro tenha chegado ao fim do período de armazenagem, ele pode ser excluído. É importante afirmar que, em alguns casos, a regulamentação exige que o registro seja completamente excluído. Além disso, registros excluídos não devem ser recuperáveis, mesmo com acesso irrestrito ao dispositivo de armazenamento. A eliminação simples dos arquivos não é suficiente para garantir que seu conteúdo não possa ser recuperado, uma vez que o conteúdo de registro, em muitos casos, pode ser reconstruído a partir das informações ainda armazenadas no disco correspondente [MW06].

Garantias na Migração de Dados. As informações contidas nos dispositivos WORM podem ser conservadas por anos. Assim, são necessários mecanismos confiáveis para transferir o conteúdo de uma mídia de armazenamento obsoleta para uma nova. Muitas regulamentações, por exemplo, exigem retenção de registros obrigatórios por décadas. No entanto, armazenar esses registros por um longo tempo exigirá mudanças tanto no *hardware* de armazenamento quanto no formato utilizado para gravar em disco. Consequentemente, a migração para novos dispositivos deve ser confiável, bem como verificável [MWHM07].

Disponibilidade e Desempenho. Os dados armazenados em dispositivos WORM devem ser acessíveis e estarem disponíveis para que sejam lidos continuamente. Em outras palavras, os dispositivos WORM devem fornecer um bom serviço a todos os seus usuários em todos os momentos. Assim, modelos de armazenamento adequados devem ser utilizados para permitir melhor desempenho, tanto de disponibilidade quanto de segurança.

Log de Auditoria. Todo acesso ao sistema de armazenamento deve ser registrado, como um meio de identificar as operações realizadas por um dado usuário. O registro de todos os dados armazenados e as operações realizadas é exigência da maioria das regulamentações atuais de controle fiscal. Com esse controle, é possível identificar o usuário que está acessando ou modificando os dados, a fim de fornecer trilhas de auditoria verificáveis.

Backup. A realização das cópias dos dados armazenados, bem como a restauração daqueles que foram indevidamente perdidos (ou corrompidos) devem ser permitidas pelo sistema WORM. As cópias devem ser distribuídas geograficamente em locais diferentes

para garantir maior disponibilidade de dados.

Baixo custo. O dispositivo deve ser barato o suficiente para que o armazenamento de grande quantidade de dados seja economicamente viável.

Espaço de armazenamento ilimitado na prática. O sistema sempre deve fornecer ao usuário espaço suficiente para o armazenamento dos dados. Substituições de dispositivos de armazenamentos devem ocorrer de forma transparente para o usuário.

2.2 SISTEMAS DE ARQUIVOS NATIVOS DE UM SISTEMA OPERACIONAL

Sistemas computacionais normalmente necessitam armazenar e recuperar dados. Devido às restrições em relação à quantidade de memória, muitas vezes é necessário que estes dados sejam armazenados fora da RAM, fazendo uso, na maioria das situações, de discos rígidos. Esses discos também são utilizados quando uma aplicação deseja que a informação manipulada seja persistente, ou seja, permaneça armazenada mesmo quando o processo encarregado pelo seu gerenciamento seja encerrado [Tan09, Mac07, Sil10].

É comum que diferentes processos de um sistema necessitem acessar o mesmo conjunto de dados ao mesmo tempo. Em boa parte das situações não é aceitável impor a restrição de que somente um desses processos possa ter acesso aos dados. Uma forma para solucionar essa questão é fazer com que os dados sejam independentes dos processos que têm acesso a eles [Tan09].

Assim, Tanenbaum [Tan09] lista como requisitos fundamentais para o problema de armazenamento de dados a longo prazo:

- a possibilidade de armazenar grande quantidade de dados;
- a necessidade de manter os dados, mesmo depois que o processo que a utilizava seja encerrado;
- o acesso concorrente ao dado, ou seja, diversos processos podem manipular os dados simultaneamente.

Uma solução eficiente para tal problema consiste na armazenagem dos dados em discos, em uma unidade denominada "arquivo". Assim, os processos podem ler dados de arquivos previamente criados ou escrever novos dados criando outros. Depois de ser criado, um arquivo deve persistir após o encerramento do processo que o criou, devendo ser removido apenas quando o seu proprietário assim desejar.

O gerenciamento dos arquivos é tarefa do sistema operacional, sendo ele o encarregado de definir as formas de organizar, de identificar, de acessar, de utilizar, de proteger e de implementar esses arquivos. A parte do sistema operacional responsável pelo tratamento dos arquivos é denominada "sistema de arquivos". Neste trabalho, designamos o termo nativo aos sistemas de arquivos implementados dentro do *kernel* do sistema operacional, como o Ext3 (Linux) [Twe00] e o NTFS (Windows) [NTF10]. Então, é função dos

sistemas de arquivos nativos: controlar o espaço disponível; determinar o tamanho dos blocos lógico; definir como os arquivos são apresentados, identificados e protegidos, etc [Tan09, Mac07, Sil10].

Um sistema de arquivos é constituído por duas partes: uma coleção de arquivos, contendo todas as informações referentes ao armazenamento de dados, e uma estruturação de diretórios, que organiza e fornece informações sobre todos os arquivos do sistema.

2.2.1 Arquivos

Arquivos são um mecanismo de abstração que propicia uma forma de armazenar e recuperar informações contidas no disco rígido. Os arquivos devem ser armazenados de forma uniforme, o que possibilita o acesso ao mesmo conteúdo por parte de diferentes processos. O armazenamento deve fornecer ao usuário uma abstração de como os dados estão sendo tratados e de como os discos operam na prática. Por convenção, as informações não podem ser escritas no disco rígido sem que estejam armazenadas dentro de arquivos.

Os sistemas de arquivos implementam uma série de operações para permitir o acesso aos arquivos por parte dos processos que rodam dentro do sistema operacional. Essas operações são responsáveis pelo armazenamento e posterior recuperação das informações gravadas no sistema. A quantidade e a funcionalidade dessas operações podem variar conforme o sistema utilizado, mas, de um modo geral, as chamadas principais podem ser resumidas a onze[Tan09]:

Create: permite a criação do arquivo sem dados, ajustando alguns dos seus atributos. Alguns sistemas não implementam tal operação, dessa forma, habilitam a criação através de flags na requisição *open()*.

Delete: remove um arquivo quando o mesmo deixa de ser necessário para o usuário. O espaço utilizado por este arquivo deve ser liberado após a execução desta operação.

Open: antes de utilizar um arquivo, o processo precisa abri-lo. Nessa operação, o sistema busca os atributos e a lista dos endereços em disco correspondentes a este arquivo, carregando estas informações na memória principal, tornando mais rápidos os acessos posteriores a estes conteúdos.

Close: fecha o arquivo, quando não houver mais acessos ao mesmo, e libera o espaço ocupado pelos seus dados na memória principal.

Read: os dados são lidos do arquivo. O processo que faz a leitura deve informar a quantidade de dados que deverão ser lidos e providenciar um *buffer* onde eles serão armazenados. Vale ressaltar que, antes da execução da operação de leitura, deve ser especificada a posição do arquivo a partir da qual a leitura será iniciada.

Write: os dados são escritos nos arquivos, a partir de uma posição previamente especificada. Se tal posição for a de final de arquivo, o mesmo aumentará de tamanho.

Caso ocorra no meio do arquivo, os dados armazenados anteriormente naquelas posições serão sobre-escritos.

Append: é uma operação de escrita que sempre adiciona conteúdo ao final do arquivo, aumentando o seu tamanho.

Seek: em sistemas que permitem acesso randômico aos dados, trata-se de uma chamada que indica o ponto a partir do qual os dados devem ser acessados, permitindo a realização de operações de leitura e de escrita.

Get Attributes: consiste no processo de leitura dos atributos dos arquivos, verificando permissões de acesso, bem como outras informações armazenadas nos metadados.

Set Attributes: consiste na modificação do conteúdo dos atributos, vale ressaltar que o sistema operacional permite que apenas alguns atributos sejam modificados pelos usuários.

Rename: modifica o nome do arquivo. Esta operação muitas vezes não é implementada, pois um arquivo pode ser copiado em outro com um novo nome, sendo o antigo removido.

A maior parte das operações mencionadas acima envolvem buscas nos diretórios associadas ao nome do arquivo. Para evitar uma quantidade demasiada de buscas, muitos sistemas necessitam que a chamada ao sistema *open()* seja executada antes do arquivo ser utilizado pela primeira vez. O sistema operacional mantém uma tabela chamada *open-file table*, contendo informações sobre todos os arquivos abertos. Então, quando uma operação sobre um arquivo é solicitada, as informações dele são carregadas a partir de um índice da tabela, sem que buscas desnecessárias voltem a ser executadas. Quando um arquivo não é mais utilizado por nenhum processo, o SO fecha-o (a partir da função *close()*), removendo sua entrada da *open-file table*.

Sistemas de arquivos precisam dar suporte ao armazenamento de quantidades elevadas de arquivos. Contudo, para gerenciar toda essa informação, é necessário organizá-la. É função dos diretórios e das partições organizar este conteúdo.

2.2.2 Diretórios

Manter os discos com apenas um tipo de sistema de arquivos é usual e aceitável para boa parte das aplicações. Contudo, certas situações necessitam alocar múltiplos *file systems* no mesmo disco rígido ou reservar partes do disco para outras finalidades, tais como abrigar o espaço para *swap*[2] no Linux. Essas partes do HD são conhecidas por partições de disco. É importante ressaltar que cada partição pode ser criada com um sistema de arquivos diferente.

[2]Sistemas operacionais modernos utilizam o disco como extensão da memória RAM. A parte do disco que é usada como memória virtual é chamada área de troca ou área de swap.

Para efeito de organização um sistema de arquivos necessita armazenar informações sobre os arquivos no sistema. Para tal, estruturas denominadas de diretórios são utilizadas, armazenando dados como o nome, a localização, o tamanho e o tipo de todos os arquivos. A Figura 2.1 apresenta um esquemático de um disco rígido dividido em duas partições. As informações sobre todos os arquivos contidos em cada partição estão armazenadas no respectivos diretórios.

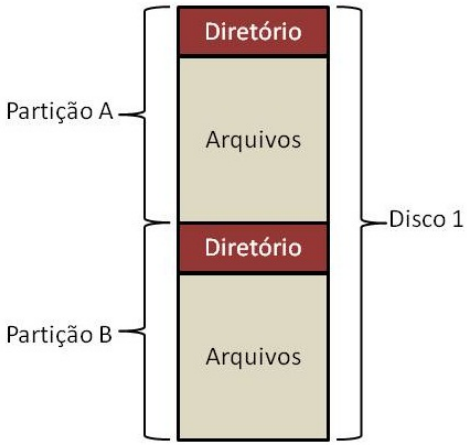

Figura 2.1: Disco rígido dividido em duas partições

Da mesma forma que os arquivos, os diretórios suportam uma série de operações. Devido à grande variação entre os sistemas operacionais disponíveis, serão detalhadas as principais chamadas executadas pelos sistemas baseados em UNIX:

CreateDir: cria um diretório vazio. Contém apenas o ponto e o pontoponto, que são colocados automaticamente pelo sistema.

DeleteDir: remove um diretório, apenas diretórios vazios podem ser apagados.

OpenDir: abre um diretório. Os diretórios podem ser lidos apenas após serem abertos.

CloseDir: fecha um diretório após sua leitura para liberar o espaço na memória.

ReadDir: retorna a próxima entrada em um diretório aberto. Apesar de ser possível ler os diretórios com a função *read()* dos arquivos, esse método tem a desvantagem de obrigar o usuário a conhecer a estrutura interna dos diretórios, enquanto o *readdir()* sempre retorna uma entrada da tabela do diretório em formato-padrão.

RenameDir: modifica o nome do diretório.

Link: técnica de ligação que permite que um arquivo apareça em mais de um diretório. Esta chamada especifica um arquivo existente e um nome de caminho, e cria uma ponte entre este arquivo e aquele especificado pelo caminho. Desta forma, o mesmo arquivo pode aparecer em muitos diretórios, sem que para isso seja necessário replicar seu conteúdo diversas vezes, apenas um atalho é criado.

Unlink: remove a entrada de um diretório. Se o arquivo a ser removido pertence somente a um diretório, ele é removido do sistema de arquivos. Se ele estiver em muitos diretórios, somente o caminho especificado é removido, os outros permanecem.

De forma semelhante a dos arquivos, que precisam ser abertos para serem utilizados, um sistema de arquivos precisa ser montado antes de estar disponível para acesso por parte dos processos do sistema. O processo de montagem é bastante simples. Inicialmente, o sistema operacional fornece o nome do dispositivo (disco rígido, volume, etc.) e o ponto de montagem, local dentro da estrutura dos arquivos onde o sistema de arquivo estará disponível para acesso. Na maioria dos casos, o ponto de montagem é formado por um diretório vazio. Após esse procedimento, é possível acessar o conteúdo do sistema de arquivos dentro do diretório representado pelo ponto de montagem. Contudo, vale ser ressaltado que, antes de relacionar o sistema de arquivos ao ponto de montagem, o sistema operacional verifica se o dispositivo passado no processo contém um sistema de arquivo válido e efetua uma varredura completa nos diretórios deste sistema a fim de verificar toda sua estruturação.

2.2.3 Implementação de Sistemas de Arquivos Nativos

Durante a implementação, os desenvolvedores de sistemas de arquivos a nível do *kernel* enfrentam pelo menos dois problemas referentes ao projeto do sistema. O primeiro é definir como o sistema de arquivos será apresentado ao usuário. Essa tarefa envolve a definição dos arquivos e dos seus atributos, das operações permitidas sobre eles, bem como da estruturação dos diretórios para organizar os arquivos. O segundo problema é a criação de algoritmos e de estruturas de dados que mapeiem as definições lógicas do sistema em soluções físicas que possam ser armazenadas e operadas pelos sistemas operacionais.

Devido à complexidade de implementação, um sistema de arquivos nativo é normalmente composto de diferentes níveis, conforme apresentado na Figura 2.2. Cada nível utiliza as funcionalidades fornecidas pelos níveis inferiores para propiciar novas funcionalidades para o nível acima nesta hierarquia.

O nível mais baixo, o controle de I/O, utiliza dos *device drivers* e dos controladores de interrupção para transferir dados entre a memória principal e o sistema de discos. Um *device driver* pode ser considerado um tradutor, visto que recebe instruções de gerenciamento dos discos em alto nível e as converte em instruções de baixo nível, que são específicas para o controle do *hardware*.

O nível logo acima, o sistema de arquivos básico, necessita apenas emitir comandos genéricos para os *device drivers* apropriados, com o objetivo de ler e escrever blocos no disco em seu endereço devido (ex.: disco 1, cilindro 73, track 2, setor 10).

O módulo de organização de arquivo tem o conhecimento sobre os arquivos, tanto seus blocos lógicos quanto os físicos. Conhecendo o tipo de alocação que foi utilizada no arquivo, este módulo pode traduzir endereços lógicos em endereços físicos, fornecendo

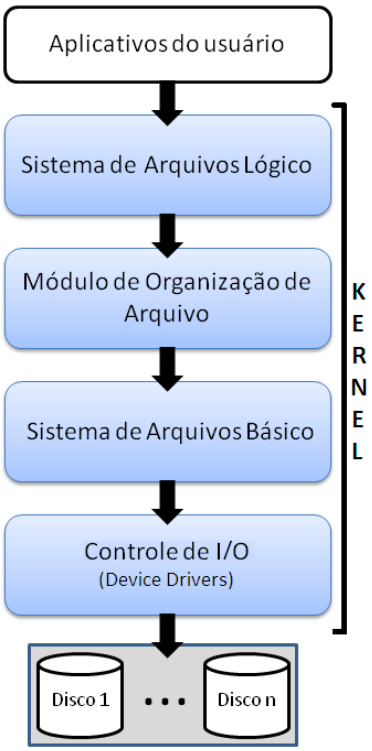

Figura 2.2: Estruturação típica de um sistema de arquivos nativo

tal informação à camada inferior. Dentre outros componentes, o módulo de organização de arquivo também inclui um gerenciador de espaço livre, que rastreia todos os blocos desalocados e os fornece ao organizador de arquivos quando solicitado.

Por último, o sistema de arquivos lógico gerencia a informação armazenada nos metadados, que, por sua vez, inclui todas as estruturas dos sistema de arquivos, exceto o conteúdo efetivo do mesmo[3]. Este módulo controla a estrutura de diretórios propiciando as informações necessárias para o módulo logo abaixo operar. O sistema de arquivos lógico armazena a estrutura do arquivo através de FCBs (*file-control-blocks*). Estrutura essa que contém informações sobre o arquivo, incluindo o dono, as permissões de acesso, a localização dos segmentos de conteúdo do arquivo. Esta camada também é responsável pela proteção e segurança do sistema de arquivos.

A criação de arquivos em disco exige que o sistema operacional tenha o controle de quais áreas ou blocos no disco estão livres. Este controle é realizado utilizando alguma estrutura de dados que armazene informações que possibilitem ao sistema de arquivos gerenciar o espaço livre em disco. A forma mais simples de implementar essa estrutura é através de uma tabela denominada de mapa de bits (*bit map*). Cada entrada da tabela é associada a um bloco do disco representado por um bit. Caso o valor de determinado bit seja igual a 0, o respectivo bloco no disco está livre, caso seja 1, o bloco está alocado.

[3]São exemplos de metadados: o nome do arquivo na forma legível pelo ser humano, a estrutura dos diretórios, as informações de acesso e de dono de arquivo, bem como os tempos de criação e de modificação do mesmo.

Assim, o mapeamento dos espaços livres pode ser realizado, bastando atualizar o mapa de bits em todas as situações que necessitem de alocação ou liberação de espaço em disco.

Gerenciar o espaço livre em disco não é suficiente, no entanto, para diminuir o risco de fragmentação, é preciso alocar eficientemente os arquivos e, em algumas situações, modificar a disposição dos arquivos em disco, procedimento conhecido por desfragmentação.

Considerando que os meios de armazenamento são compartilhados entre diversos usuários, é de fundamental importância que mecanismos de proteção sejam implementados para garantir a proteção individual de arquivos e de diretórios. Então, qualquer sistema de arquivos deve possuir mecanismos próprios para proteger o acesso às informações gravadas em disco, além de possibilitar o compartilhamento de arquivos entre usuários, quando desejado.

Como o acesso ao disco é muito mais lento que o acesso à memória, sistemas de arquivos nativos devem se preocupar em otimizar o desempenho das suas operações. A armazenagem de blocos em *cache*, a leitura antecipada e a redução do movimento do braço do disco são as principais técnicas para melhorar o desempenho nesses sistemas. Contudo, como não é o foco deste trabalho desenvolver soluções em sistemas de arquivos nativos, não será abordada, em mais detalhes, a implementação destes.

2.3 LINUX: MODO DE OPERAÇÃO E SISTEMAS DE ARQUIVOS

Um sistema Linux pode ser considerado um tipo de pirâmide, como ilustrado na Figura 2.3. Na base dessa pirâmide está o *hardware*. Executando diretamente sobre estes dispositivos está o sistema operacional Linux. Esta camada é conhecida como núcleo do sistema (*kernel*), é desenvolvida na Linguagem C e faz uso de estrutura de dados complexas. Sua função é controlar o *hardware* e fornecer uma interface de chamadas de sistema para todos os programas, permitindo que os programas dos usuários criem e gerenciem processos, arquivos e outros recursos. Quando uma atividade do núcleo do SO está em execução em um dado momento na CPU, o Linux considera-se executando em modo *kernel* ou modo núcleo. No entanto, quando utilitários ou programas do usuário estão executando funções da biblioteca do sistema operacional, é dito que o Linux está trabalhando em modo usuário. Vale ressaltar que o Linux define interfaces claras de comunicação entre todas as camadas da pirâmide [Tan09, Mac07, Sil10].

Além do sistema operacional e da biblioteca de chamadas ao sistema, todas as versões de Linux fornecem um grande número de programas-padrão[4], alguns dos quais são especificados pelo padrão POSIX 1003.2 [Kir10], tais como shell, compiladores, editores, programas de processamento de texto e utilitários de manipulação de arquivos.

Um arquivo do Linux é uma sequência de 0 ou mais bytes contendo qualquer dado. Nenhuma distinção é feita entre os tipos de arquivos. Os arquivos são agrupados em diretórios por questões de conveniência, e estes, por sua vez, armazenados como arquivos.

[4]Estes são os programas que um usuário executa em um terminal Linux, independentemente de qual distribuição esteja utilizando.

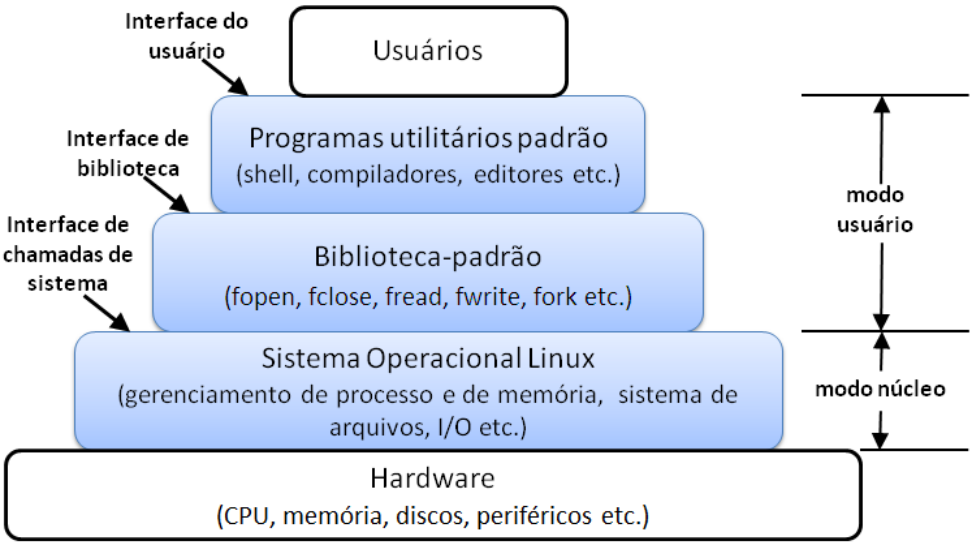

Figura 2.3: Modos de operação Linux

A Figura 2.4 representa a estruturação dos diretórios no Linux. Esses diretórios podem ser subdivididos em diversos níveis, sendo interligados por apontadores para as outras estruturas a partir do diretório-raiz [Tan09].

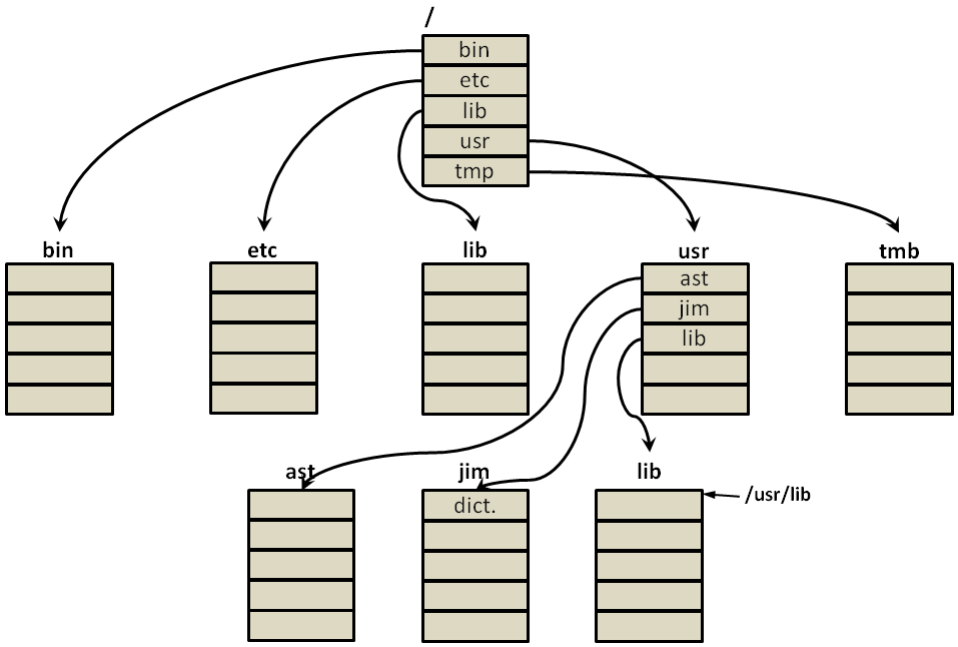

Figura 2.4: Estrutura de um diretório no Linux

O Linux permite o acesso a cada dispositivo considerando-os arquivos especiais no sistema, o que possibilita, dessa forma, a integração com os sistemas de arquivos. Os

dispositivos de I/O, por exemplo, possuem uma regra fixa para nomenclatura[5], associada a um nome de caminho, geralmente em */dev*. Assim, esses arquivos especiais podem ser acessados da mesma maneira que os demais arquivos.

No Linux, a chamada (*close()*) não salva o conteúdo em disco, simplesmente libera o espaço em RAM o qual o SO estava utilizando para armazenar a *metadata* do arquivo (fazendo a associação entre o número do arquivo e o seu nome).

2.3.1 Virtual File System

Para garantir homogeneidade entre as operações dos sistemas de arquivos, o *kernel* do Linux implementa o conceito de *Virtual File System* (VFS), um mecanismo que permite às chamadas de sistemas genéricas, tais como *open()* e *read()*, serem executadas independentemente do sistema de arquivos usado ou do meio físico [Tan09].

O VFS, também conhecido como *Virtual File Switch*, é um sub-sistema do *kernel* que implementa as interfaces para que os programas dos usuários acessem o sistema de arquivos. Conforme exposto na Figura 2.5, o VFS define tanto a interface que assegura a interação entre os programas dos usuários com o sistema de arquivos (POSIX), quanto a interface que deve ser implementada pelo sistema de arquivos para que os VFS acesse suas funcionalidades. Dessa forma, o processo do usuário, ao solicitar uma operação em um sistema de arquivos, está, na verdade, gerando uma requisição ao VFS. O *Virtual File System* verifica qual dos sistemas de arquivos é o responsável pela partição onde a operação foi solicitada, chamando as operações deste sistema necessárias para resolver tal requisição [Tan09].

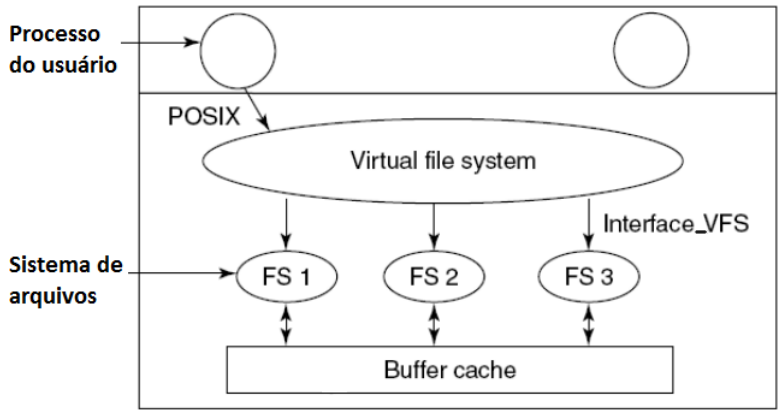

Figura 2.5: Estrutura do VFS [Tan09]

A partir desta padronização, todos os sistemas de arquivos podem co-existir e interagir

[5]O /dev/sda1 representa a primeira partição do primeiro disco SCSI, o /dev/sdb3 representa a terceira partição do segundo disco SCSI, etc.

entre si. Vale ser ressaltado que a utilização do VFS permite que mais de 50 sistemas de arquivos sejam, oficialmente, suportados pelo *kernel*. O VFS provê um modelo comum capaz de representar todas as características genéricas de um sistema de arquivos, além de definir um conjunto básico de conceitos de interfaces e estruturas de dados que todos os sistemas de arquivos suportem, tais como arquivos, diretórios, operações de criação e de remoção, etc [Tan09].

O código real do sistema de arquivos oculta os detalhes da implementação. Para a camada VFS e para o resto do Kernel, todos os sistemas de arquivos parecem iguais. O VFS é orientado à objetos, mas, como o kernel é programado em C, as estruturas de dados são simuladas nesta linguagem, contendo dados e ponteiros para funções do sistema de arquivos que operam sobre os dados.

O VFS define uma estrutura, denominada *file_operations*, onde estão definidas as operações que podem ser chamadas a qualquer sistema de arquivos Linux por processos do usuário. Essas chamadas possuem uma padronização, sendo, portanto, uma interface bem-definida para os sistemas de arquivos. No entanto, nem todas as operações precisam ser implementadas por um sistema de arquivos, às consideradas desnecessárias podem ser atribuídos os valores NULL.

2.3.2 Third Extended File System (Ext3)

Um dos sistemas de arquivos mais difundidos nas distribuições Linux é o Ext3 (*Third Extended File System*), adotado como padrão por importantes distribuições Linux até o ano de 2010, tais como o Debian Lenny e Ubuntu 9.04. O Ext3 é uma evolução, lançada em 1999, do sistema de arquivos Ext2, possuindo compatibilidade total com este sistema, pois, basicamente, adiciona a propriedade de "journal" à sua versão anterior.

O *Journaling* é a capacidade de um sistema de arquivos de acompanhar as mudanças que serão realizadas nele antes que realmente sejam feitas (por exemplo, gravações ou atualizações de dados). Essas informações capturadas pelo *Journaling* são, então, armazenadas em uma parte separada do sistema de arquivos, denominada *Journal* (também denominada de "registros de log"). Após os registros serem efetivamente armazenados no *Journal*, o sistema de arquivos aplica as devidas mudanças nos arquivos e, então, remove as informações do *Journal* caso a operação tenha sido bem sucedida. Dessa forma, se o computador for indevidamente desligado, o processo de montagem do Ext3 no próximo *startup* verificará que há mudanças descritas no Journal que não foram aplicadas. Caso existam, essas mudanças são, então, aplicadas ao sistema de arquivos. Isso faz com que os riscos de perda de dados sejam reduzidos drasticamente, sem que a interferência do usuário seja necessária.

Segundo [Evi04], as operações efetuadas nos arquivos são registradas nessa área de log do mesmo modo que o controle de transações em bancos de dados. As operações são primeiramente gravadas no *journal*. Quando a atualização do registro de ações (log) é completada, um registro de complemento (*commit record*) é gravado sinalizando o final da entrada. Então, as mudanças são efetivamente gravadas em disco no sistema de arquivos

normal.

Vale salientar que o Ext3 suporta três diferentes modos de trabalho para seu mecanismo de *Journaling* [Evi04] e [Inf10]:

Journaling: grava todas as mudanças em sistema de arquivos e usa um arquivo maior para registros de ações, o que pode retardar a recuperação durante a reinicialização. Esse é o mais lento dos três modos, por outro lado, é o que possui maior capacidade de evitar perdas de dados. Uma forma de aperfeiçoar a velocidade dessa opção é gravar os registros em bancos de dados externos.

Ordered: armazena somente mudanças em arquivos metadados (arquivos que possuem informações sobre outros arquivos), mas registra as atualizações no arquivo de dados antes de fazer as mudanças associadas ao sistema de arquivos. Este *Journaling* é o padrão nos sistemas de arquivos Ext3, sendo a melhor opção para a maioria dos sistemas.

Writeback: também só grava mudanças para o sistema de arquivo em metadados, mas utiliza o processo de escrita do sistema de arquivos em uso para gravação. É o mais rápido *Journaling* Ext3, porém é o menos confiável e mais suscetível à corrupção de arquivos após uma queda do sistema. Esse modo é equivalente à instalação de um sistema com EXT2 nativo.

2.4 SISTEMAS DE ARQUIVOS NO ESPAÇO DO USUÁRIO

Implementar um sistema de arquivos nativo para SOs baseados em Unix (como o Linux) é uma tarefa reconhecidamente difícil [Tan09, Sil10], pois exige do desenvolvedor um profundo conhecimento do código do *kernel*, bem como das estruturas de dados utilizadas pelo sistema operacional. Vale ressaltar que políticas de acesso e de proteção à memória devem ser implementadas, assim como cuidados com a sincronização de primitivas do SO devem ser tomados, o código deve ser escrito apenas em C e possui um processo de depuração extremamente longo, necessitando reiniciar o sistema em caso de erro. Devido à essa complexidade, tanto a curva de aprendizagem quanto o tempo de desenvolvimento de FS nativos costumam ser mais longos do que o usual para desenvolver os sistemas implementados no espaço do usuário. Além disso, garantir que o código-fonte desenvolvido esteja livre de erros que ocasionem perdas de informação é uma tarefa extremamente dura, devido ao tamanho e à complexidade do código do sistema nativo.

Sistemas de arquivos nativos possuem desvantagens, pois dificilmente podem ser migrados para outros sistemas operacionais sem que modificações significativas no projeto e na implementação sejam realizadas, mesmo quando possuem interfaces de sistemas de arquivos similares (como a camada de VFS). Outro ponto negativo é que um sistema de arquivos no espaço do *kernel* necessita que o usuário possua privilégios de administrador para conseguir montá-lo.

A programação no espaço do usuário consiste na implementação de sistemas exter-

namente ao núcleo do SO, diferentemente do desenvolvimento no espaço do *kernel* do sistema operacional. Dessa forma, a maior parte das restrições mencionadas anteriormente são eliminadas. Logo, geralmente não é necessário escrever módulos do *kernel* quando os sistemas de arquivos são implementados desta forma, isentando o desenvolvedor de conhecer profundamente o funcionamento das estruturas de dados e do código do *kernel* para desenvolver um sistema de arquivos.

Sistemas de arquivos no espaço do usuário propiciam um conjunto de benefícios aos desenvolvedores e aos usuários, como a facilidade de desenvolvimento, a portabilidade entre sistemas operacionais e a possibilidade de utilização por parte de usuários sem privilégios de administrador. Para muitas aplicações, essas vantagens superam o *overhead* de desempenho quando comparado aos sistemas de arquivos nativos do sistema operacional. Essa perda de desempenho ocorre porque eles definem o comportamento das operações do sistema de arquivos (acesso, leitura, escrita etc.), mas não são diretamente responsáveis pelas operações de baixo nível realizadas. Sistemas de arquivos no espaço do usuário definem como e onde as informações devem ser armazenadas/acessadas, controlam as tarefas a serem executadas, mas, no entanto, deixam ao encargo de *frameworks* como o FUSE e do sistemas de arquivos nativos a execução das operações de baixo nível, como gerenciamento de memória, escrita e leitura dos setores do disco rígido, etc. Tais *frameworks*, por sua vez, utilizam-se de algumas funcionalidades propiciadas pelo sistema de arquivo nativo do SO para realizar as suas tarefas [Tan09, RG10].

O FUSE é um framework para sistemas baseados em UNIX bastante difundido entre os desenvolvedores Linux. Ele oferece uma infraestrutura que permite a implementação de sistemas de arquivos no espaço do usuário. Além disso, garante: portabilidade entre diversos sistemas operacionais, utilização por usuários sem privilégios de administrador, além de abstração das operações de baixo nível. A Seção 2.4.1 discorre sobre o *Filesystem in Userspace*.

2.4.1 FUSE: Filesystem in Userspace

O FUSE oferece aos desenvolvedores uma API consistindo de operações tradicionais de sistema de arquivos, além de suporte a diversas linguagens de programação, tais como C, Java e Python. Sistemas baseados em FUSE podem ser montados por usuários sem privilégios de administrador de forma segura. Outra vantagem de desenvolver sistemas de arquivos com o FUSE é o fato de não ser necessário recompilar ou reiniciar o *kernel* do sistema operacional, nem durante o desenvolvimento, nem durante a instalação do sistema de arquivos [RG10, Sze09]. Facilidades como essas, associadas à portabilidade, à abstração as operações de baixo nível, motivam a utilização de sistemas de arquivos virtuais em aplicações inovadoras, como por exemplo o WikipediaFS [Blo07], que permite aos usuários visualizar e modificar artigos da Wikipedia como se estes fossem arquivos locais.

O Linux inclui oficialmente os pacotes do FUSE nas versões do *kernel* iguais ou superiores à 2.6.14 [FUS10a]. Outros sistemas operacionais, como MAC OS X [Sin10a],

Windows [FUS10b], OpenSolaris [Sol10], FreeBSD [Hen10] e NetBSD [Net10], também apresentam módulos que suportam o FUSE que podem ser instalados pelos usuários. Vale ser ressaltado que um dos motivos do sucesso do FUSE é a flexibilidade da sua licença, visto que ele é disponibilizado sobre uma licença que permite tanto o desenvolvimento de sistemas de arquivos livres, quanto de sistemas comerciais ou proprietários.

O FUSE é constituído por um módulo localizado no *kernel* do SO (*fuse.ko*), uma biblioteca no espaço do usuário (*libfuse*) e um utilitário de montagem (*fusermount*). O Módulo do *kernel* é o responsável pela conexão do sistema de arquivos com o VFS, localizado no núcleo do SO. A *libfuse* é uma biblioteca no espaço do usuário responsável pela comunicação com o módulo do *kernel*. Enquanto o *fusermount*, por sua vez, é necessário devido ao fato de que a montagem de sistemas de arquivos, em ambientes Linux, é uma operação restrita aos usuários com privilégios de administração, logo este módulo é o responsável por assegurar que usuários, mesmo sem privilégios, poderão montar sistemas baseados em FUSE. Este utilitário também propicia aos usuários uma série de parâmetros de configuração, tais parâmetros foram importantes para o estudo de desempenho realizado no Capítulo 4.

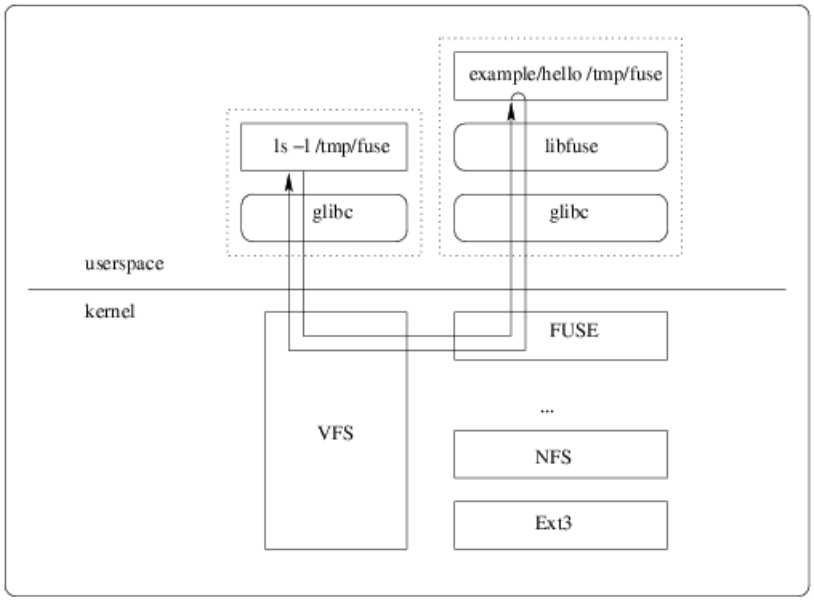

Figura 2.6: Funcionamento do FUSE no Linux [Sze10]

A Figura 2.6 apresenta o funcionamento do FUSE respondendo a uma requisição do sistema. A chamada é destinada primeiramente ao *kernel* pela aplicação, neste caso a *ls*. Após esta ocorrência, o VFS repassa a requisição para o módulo do FUSE localizado no *kernel*, que, por sua vez, repassa esta solicitação à biblioteca do FUSE no espaço do usuário. Então, a biblioteca do FUSE executa a função do *userspace file system* adequada para resolver tal requisição. O FUSE passa a controlar, portanto, qual a função do sistema de arquivos deve ser executada. O módulo do FUSE localizado no *kernel* se comunica com a biblioteca FUSE do espaço do usuário, através de um descritor de arquivo especial que é obtido ao abrir o */dev/fuse*. Este arquivo pode ser aberto diversas vezes e o descritor

de arquivo obtido é passado para a *syscall* responsável pela montagem do sistema, para associar o descritor com o sistema de arquivos montado.

Um sistema de arquivos fictício, doravante chamado de fusefs, será utilizado neste trabalho, com o objetivo de descrever, em mais detalhes, o comportamento do FUSE. Antes de iniciarmos a descrição do funcionamento deste *framework* é importante informar que o fusefs levado em consideração implementa a API de alto-nível do FUSE a partir da linguagem C. Logo, as funções contidas no arquivo *fusefs.c* responsáveis por definir o comportamento do fusefs podem ser acessadas pelo FUSE por meio de ponteiros de função (*callbacks*).

A partir do momento que uma partição gerenciada pelo fusefs é montada, todas as chamadas ao sistema de arquivos são repassadas para o módulo do FUSE localizado no kernel. Este módulo faz a associação, no VFS, entre o fusefs e a partição que acaba de ser passada como argumento no processo de montagem. A Figura 2.7 apresenta um exemplo da execução de uma operação de leitura de arquivo através do FUSE. Neste caso, o diretório */fuse* (localizado no sistema de arquivos nativo) é escolhido como o ponto de montagem do fusefs. Quando uma aplicação solicitar a operação de leitura, *read()*, para o arquivo */fuse/file*, o VFS executa o controlador do fusefs. Se o controlador identificar que os dados solicitados já estão armazenados na *page cache*, eles serão retornados imediatamente. Caso contrário, a chamada ao sistema é repassada para a biblioteca *libfuse* que, por sua vez, retorna a implementação da função *read()* localizada no fusefs. Esta função é, então, posta em execução e retorna, no *buffer* passado como argumento, os dados desejados após a leitura da informação [RG10].

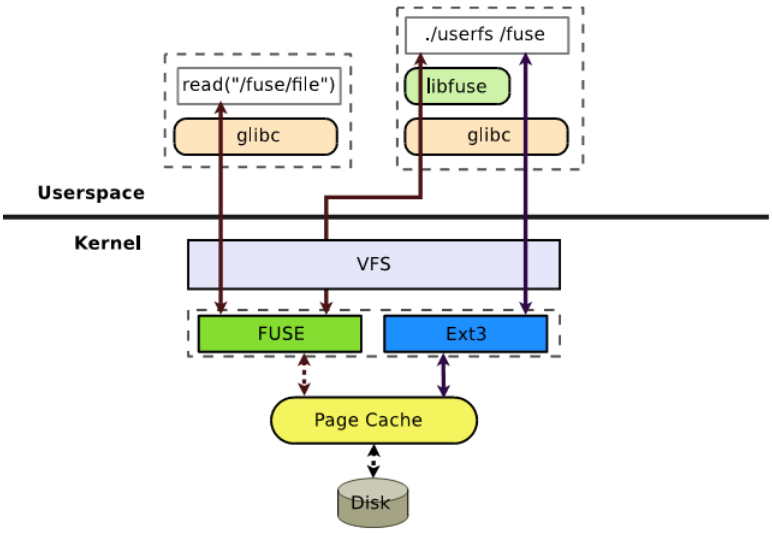

Figura 2.7: Funcionamento de uma operação de leitura em um sistema de arquivos típico baseado no FUSE [RG10]

Por se tratarem de sistemas de arquivos virtuais, os *file systems* desenvolvidos a partir do FUSE apresentam alguma perda de desempenho, quando comparados com os nativos do sistema operacional. Quando operações são realizadas utilizando apenas o sistema

de arquivos nativo (ex. Ext3 FS) não há a necessidade de efetuar trocas de contexto entre processos. No entanto, ocorrem duas trocas de modos de operação: modo *kernel* para modo usuário e vice-versa. Essa operação é muito menos custosa que as troca de contexto. Contudo, ao utilizar o FUSE, duas trocas de contexto e duas de modo de operação são realizadas. O FUSE introduz trocas de contexto para cada operação (*system call*) realizada, devido à necessidade, por parte do processador, de escalonar entre a aplicação do usuário e a biblioteca do FUSE, localizada dentro do espaço do usuário. A outra troca de contexto ocorre no sentido inverso [RG10].

O FUSE também introduz duas cópias adicionais na memória para argumentos de entrada e de saída do sistema de arquivo desenvolvido. Ou seja, esse *overhead* atinge os *buffers* contendo os dados lidos ou escritos pelo sistema de arquivos. Quando apenas o sistema de arquivos nativo é utilizado, o dado precisa ser copiado na memória apenas uma vez. Os dados que, por sua vez, serão escritos por um sistema de arquivos FUSE precisam, primeiramente, serem copiados da aplicação para a *page cache*, então da *page cache* para o *libfuse* via */dev/fuse* e, finalmente, da *libfuse* para a *page cache*, quando a operação é repassada ao sistema de arquivo nativo. Uma sobrecarga similar pode ser observada em operações de leitura. Ao utilizar a opção de montagem *Direct I/O*, o módulo do FUSE localizado no kernel do sistema operacional desabilita a escrita na *page cache*, reduzindo o número de cópias adicionais na memória para um. Dessa forma, as operações de escrita são mais rápidas, porém, o desempenho de leitura pode sofrer impacto negativo. Embora exista uma perda de desempenho por parte dos sistemas FUSE quando comparados com sistemas de arquivos nativos, alguns trabalhos indicam que, dependendo da carga utilizada, esta perda pode ser suficientemente leve[6] a ponto de não impactar negativamente na utilização de sistemas virtuais por usuários residenciais [RG10].

O desenvolvedor de sistemas de arquivos no espaço do usuário pode optar por um dos dois tipos de API propiciadas pelo FUSE. A API de baixo nível é bastante parecida com a interface do VFS original, dessa forma, o sistema de arquivo desenvolvido necessita gerenciar os inodes, bem como converter os caminhos dos arquivos armazenados em disco. Adicionalmente, o desenvolvedor deve gerenciar as estruturas de dados utilizadas para comunicação com o *kernel*. Por outro lado, a API de alto nível do FUSE propicia uma série de benefícios, visto que o desenvolvedor não precisa gerenciar os inodes e trabalha com um conjunto simplificado de instruções. A *libfuse* é encarregada de executar e de armazenar em *cache* as operações sobre os inodes, assim como de gerenciar as estruturas de dados para comunicação com o *kernel*, fornecendo abstração ao desenvolvedor que utilize tal API. Vale ressaltar que a maior parte dos sistemas de arquivos desenvolvidos a partir do FUSE podem ser implementados com base na API de alto nível [RG10].

[6]Quando o sistema de arquivo for implementado utilizando a Linguagem C. Arquivos em Java apresentam maior perda de desempenho.

2.5 AVALIAÇÃO DE DESEMPENHO DE SISTEMAS

Um dos principais objetivos da Avaliação de Desempenho de Sistemas (ADS) é obter ou fornecer o melhor desempenho ao menor custo para sistemas computacionais [Jai08]. Como resultado desse esforço, temos evolução no desempenho e na eficiência de estações de trabalho e de computadores pessoais. Com o amadurecimento da área de projeto de computadores, a indústria de tecnologia está se tornando mais competitiva e a utilização de técnicas de ADS pode ser um importante meio para garantir que a alternativa selecionada ofereça a melhor relação custo-desempenho [Jai08].

A avaliação do desempenho é necessária em todas as fases do ciclo de vida de um sistema computacional. A ADS deve ser utilizada quando o projetista do sistema deseja comparar uma série de projetos alternativos e encontrar o melhor. Mesmo nos casos em que não há opções distintas a serem comparadas, a avaliação de desempenho contribui para determinar quão bem o sistema atual está realizando as atividades e indicar se alguma melhoria precisa ser feita. Normalmente, os cenários de avaliação dentro do universo das aplicações computacionais são tão numerosos que não é possível estabelecer uma medida padrão de desempenho. Em muitos casos, também não é possível especificar um ambiente de medição padrão, nem mesmo uma técnica padrão para todos os casos. Vale ressaltar que o primeiro passo na avaliação de desempenho é selecionar as métricas certas, bem como um ambiente de medição que supra as necessidades (incluindo ferramentas de medição) e as técnicas corretas para análise [Jai08].

Antes de iniciar o monitoramento do sistema, as métricas de desempenho devem ser definidas. Os parâmetros dessas métricas são estabelecidos de acordo com várias condições, em diferentes períodos do dia, semana, mês ou, até mesmo, quando o *hardmare/software* são alterados. Vale destacar que, segundo Lilja [Lil05], uma métrica deve possuir linearidade, confiança, repetitividade, consistência, independência e facilidade de medição.

Outro ponto importante na monitoração de desempenho é a detecção de gargalos (*bottlenecks*). Esses gargalos podem ocorrer por diversos motivos, como, por exemplo, ambientes configurados de forma inadequada e recursos defeituosos ou insuficientes para a demanda real de utilização. Esses problemas são facilmente diagnosticados através de uma criteriosa avaliação de desempenho.

Através da monitoração e da análise de dados é possível identificar quais recursos são mais utilizados, além de verificar se eles são os responsáveis pela redução de desempenho do sistema. Memória, interface de rede, processador, processo e disco são alguns dos subsistemas que podem ser monitorados.

Este trabalho utilizou a estratégia de medição por amostragem. Nessa abordagem, o registro de infomações do sistema ocorre em intervalos de tempo fixos. Os resultados obtidos a partir das amostras são um resumo estatístico do comportamento global do sistema. No entanto, eventos que ocorrem com baixa frequência podem ser perdidos, devido à aproximação estatística utilizada [Lil05]. Mais detalhes sobre o procedimento de medição adotado neste trabalho podem ser encontrados na Seção 4.1

Como as ferramentas de monitoração de desempenho do sistema normalmente executam no próprio ambiente de medição, é inevitável que haja uma modificação de comportamento do que está sendo medido. Quanto maior a quantidade de informações e a resolução da medição fornecida pela ferramenta, maior será a perturbação introduzida por ela. Essa perturbação torna os dados coletados menos confiáveis. Para evitar que essas perturbações tragam inconsistências relevantes às amostras, as ferramentas de monitoração e os intervalos de amostragem devem ser criteriosamente definidos.

Com o objetivo de evitar erros durante a realização dos experimentos, assim como o de adotar técnicas corretas e eficientes de mensurar e comparar o desempenho do WFS, fontes sobre Avaliação de Desempenho de Sistemas foram consultadas [Jai08, ET97, FM03, Gen94, Lil05]. As técnicas mais importantes para este trabalho são: experimento fatorial completo [Jai08], teste de normalidade [FM03, Lil05], teste t [FM03, Lil05] e *bootstrap* [ET97].

Em alguns estudos de avaliação de desempenho, diferentes cenários de um mesmo sistema precisam ser confrontados uns com os outros com o objetivo de identificar as causas de possíveis diferenças de desempenho. A técnica de experimento fatorial completo [Jai08] consiste em comparar o desempenho do sistema sob diversos cenários de teste. Esses cenários são obtidos ao variar os valores dos fatores (parâmetros de configuração) dentro de amplitudes controladas. Como resultado, essa técnica fornece a relevância de cada fator na modificação do desempenho do sistema.

Para assegurar que as amostras coletadas se comportem, estatisticamente, como o sistema real, algumas técnicas estatísticas precisam ser aplicadas às amostras. O teste t pode ser utilizado para comparar o desempenho de dois sistemas, pois realiza análises estatísticas em suas amostras. A partir dessas comparações é possível afirmar se existe diferença estatisticamente significativa entre as amostras e determinar qual sistema possui melhor desempenho. No entanto, para que o teste t seja representativo, é necessário que as amostras se aproximem da distribuição gaussiana. Essa verificação pode ser realizada através do teste de normalidade [FM03, Lil05].

Outra técnica de Avaliação de Desempenho de Sistemas aplicada neste trabalho é o *bootstrap*, método computacional de reamostragem voltado para inferência estatística. Com *bootstrap* é possível estimar erros e calcular intervalos de confiança da amostra. Após a aplicação do *bootstrap*, os valores obtidos normalmente se aproximam da curva gaussiana. Dessa forma, é possível realizar comparações entre as amostras utilizando, por exemplo, o teste t. É importante destacar que o método *bootstrap* é mais flexível do que os métodos clássicos que podem ser analiticamente intratáveis ou aplicáveis em determinadas situações [ET97].

CAPÍTULO 3

WRITE-ONCE READ-MANY FILE SYSTEM

O presente capítulo discorre sobre o WFS (*Write-Once Read-Many File System*), um sistema de arquivos no espaço do usuário baseado na política WORM. Ao longo das seções, são apresentados uma visão geral, os requisitos e os detalhes de implementação do sistema proposto nesse trabalho.

3.1 CLIENTE E REQUISITOS

A *EMC Corporation*, empresa norte-americana de destaque internacional no ramo de dispositivos de armazenamento de dados, firmou o primeiro projeto de cooperação com a Universidade Federal de Pernambuco, coordenado pelo Professor Dr. Paulo R. M. Maciel (Centro de Informática) no primeiro semestre do ano de 2009. O acordo tinha por objetivo principal desenvolver uma solução baseada nas características *Write-Once Read-Many* voltada para os usuários residenciais Linux.

O projeto de colaboração foi definido para ser desenvolvido em três fases: a primeira teve o foco na pesquisa das características WORM e das soluções já existentes na literatura e/ou no mercado; a segunda consistiu na prototipação de um sistema de arquivos com funcionalidades *Write-Once Read-Many*, e, por fim, uma avaliação de desempenho do sistema desenvolvido foi realizada. A partir das reuniões semanais com integrantes da empresa americana, os requisitos do sistema de arquivos foram elicitados.

3.1.1 Requisitos do Sistema

Este trabalho propõe aos usuários de distribuições Linux um sistema de arquivos a partir do qual remoções ou alterações involuntárias em arquivos sejam evitadas. Contudo, essas restrições não devem afetar a criação de conteúdo, portanto, a propriedade da imutabilidade deve ser associadas aos arquivos e diretórios logo após sua criação. Para atender as necessidades do cliente, o sistema deve possuir as seguintes características:

Características WORM: o sistema de arquivos desenvolvido deve propiciar a seus usuários algumas características dos *Write-Once Read-Many*, permitindo que o conteúdo seja criado, mas impedindo as operações de remoção e de modificação sejam executadas na região de atuação do sistema;

Código-livre: o sistema desenvolvido deve ser uma ferramenta de código-livre e de utilização gratuita para usuários residenciais de distribuições baseadas em Linux;

Usuários sem privilégios: a solução não deve requisitar que os usuários possuam privilégios de administração (*root*) do sistema para ser instalada e utilizada;

Facilidade de instalação e de configuração: o sistema deve ser instalado, configurado e utilizado pelos seus usuários de forma simples. O processo de instalação deve evitar a inserção de módulos no núcleo do SO, pois muitas vezes essas operações são restritas aos administradores do sistema;

Desempenho: A solução deve apresentar desempenho similar ao Ext3 não-WORM. Nesse trabalho, foi determinado que a diferença de desempenho entre os dois sistemas de arquivos, para os cenários de teste avaliados, não deveria superar 20%;

Modo alternativo de modificação e de remoção: embora a política WORM não permita a remoção e a modificação dos dados, o sistema desenvolvido deve propiciar um modo a partir do qual os usuários possam realizar alterações no conteúdo já criado. Contudo, não deve ser possível acessar esse recurso a partir do diretório no qual as restrições WORM estão aplicadas;

Simplicidade no gerenciamento dos dados: o sistema deve propiciar um gerenciamento de arquivos de forma simples. Com o objetivo de simplificar a manutenção de arquivos, as tarefas de gerenciamento de permissões e de donos dos conteúdos devem ser desabilitadas. Os dados devem sempre estar disponíveis para leitura. Também não devem haver restrições quanto a criação arquivos novos [1].

A partir desses requisitos, um sistema de arquivos no espaço do usuário foi implementado. No entanto, vale destacar que, por se tratar de uma ferramenta voltada para usuários domésticos, decidiu-se por não incluir, entre os requisitos do sistema, a proteção contra ataques maliciosos, evitando, assim, o aumento da complexidade do código e a perda de desempenho em relação aos sistemas não-WORM.

3.2 WFS: UMA VISÃO GERAL

O WFS [Fal10, FAM$^+$10a, FAM$^+$10b] é um sistema de arquivos com características WORM desenvolvido a partir da infraestrutura FUSE (*Filesystem in Userspace*) [Sze10]. O sistema proposto nesse trabalho atende às necessidades de usuários comuns Linux, sem prioridade de administrador no sistema, que precisam reduzir o risco de apagar ou de modificar involuntariamente o conteúdo de arquivos. Vale ser ressaltado que, para utilizar-se do WFS, não é necessário que o usuário adquira ou modifique seu computador pessoal.

Por ser um sistema de arquivos no espaço do usuário baseado em FUSE, o WFS propicia a portabilidade que é característica desses sistemas. Assim, usuários de todos os sistemas operacionais que possuem versões disponíveis do FUSE podem usufruir da proteção WORM, visto que o WFS necessita apenas da instalação do FUSE no ambiente

[1] Contudo os nomes devem ser diferentes dos arquivos já existentes.

para funcionar. Como muitas das distribuições Linux atuais já trazem o FUSE incorporado ao sistema padrão, os usuários dessas distribuições não precisam ter privilégios de superusuário para usufruir das características WORM. Além disso, é comum que os administradores optem por instalar os pacotes do FUSE mesmo nas distribuições que não os inserem por padrão, uma vez que sistemas de arquivos como NTFS-3G [Sza10] e EncFS [Gou10], amplamente difundidos na comunidade Linux, são baseados em FUSE.

O código do WFS é distribuído de forma livre [Fal10] para utilização e modificação, detalhes da licença de utilização encontram-se no Apêndice A. Seu processo de instalação e de configuração consiste de dois comandos a serem executados no console da distribuição Linux utilizada pelo usuário, como detalhado na Figura 3.1. O primeiro comando é destinado a compilar o código-fonte do WFS, utilizando o GCC (compilador da linguagem C) para criar o código executável do sistema de arquivo. Após a etapa de compilação, o comando *wfs* é utilizado para montar o WFS em um diretório a partir do qual os dados protegidos poderão ser acessados. A opção de montagem Direct I/O deve ser habilitada, uma vez que os estudos apresentados no Capítulo 4 relatam um relevante ganho de desempenho ao utilizá-la. Além da opção do Direct I/O, dois diretórios devem ser passados como argumentos no processo de montagem, o primeiro deve ser a pasta não-WORM onde se encontram os conteúdos que devem ser protegidos pelo WFS, o segundo é a pasta onde os arquivos poderão ser acessados com a proteção WORM, doravante chamado de *diretório-WORM*. Assim, os dados estarão protegidos sempre que acessados diretamente pela pasta WORM. Logo, os usuários devem direcionar seus acessos e suas operações a esse diretório. Como faz parte dos requisitos do sistema, existe uma forma de modificar ou apagar os dados que, por ventura, tenham sido criados erroneamente pelo usuário, acessando os arquivos através da pasta não-WORM.

```
> gcc -o wfs -Wall -ansi -W -std=c99 -g -ggdb -D_GNU_SOURCE
 -D_FILE_OFFSET_BITS=64 -lfuse WFS.c

> ./wfs -o direct_io readwrite_directory WORM_mount_point
```

Figura 3.1: Descrição dos comandos para compilar e montar o WFS, respectivamente.

O WFS pretende propiciar aos seus usuários uma forma de evitar o risco de remoções e modificações indesejáveis de conteúdo através dos princípios do *Write-Once Read-Many*. Como principais funcionalidades WORM suportadas pelo WFS, podemos citar:

- Os arquivos e os diretórios podem ser criados sem restrições na região do disco rígido onde se encontra montado o WFS. Além disso, os usuários têm acesso irrestrito à leitura de conteúdo dentro do diretório protegido. Contudo, as modificações, as remoções e as renomeações de arquivos estão proibidas.
- Os usuários têm a permissão de copiar e/ou modificar o conteúdo dentro do diretório-WORM apenas se os arquivos forem salvos com nomes diferentes dos originais. Vale reforçar que operações de sobre-escrita também estão proibidas.

Posteriormente, um mecanismo de *trace* foi integrado ao WFS. Quando habilitado, tal mecanismo cria um arquivo de log com o histórico da execução do WFS, informando a ordem em que as funcionalidades do sistema foram executadas pelo FUSE. Parâmetros como o número de *bytes* escritos por operação de escrita podem ser adicionados, assim como mensagens para ajudar na depuração de erros. A partir das informações obtidas através desse mecanismo, foi possível conhecer o comportamento do FUSE e do WFS. O *trace* também propiciou identificar e solucionar gargalos nas operações de leitura e escrita, aprimorando o desempenho do WFS.

O WFS implementa todas as funções definidas na API de alto-nível propiciada pelo FUSE. Contudo, ao tentar executar operações como remoção, renomeação ou modificação de conteúdo, um erro de falta de permissão é lançado no sistema operacional, que, por sua vez, repassa tal mensagem ao usuário. Assim, os arquivos e diretórios podem ser criados usando os comandos originais do Linux caso o arquivo ou diretório de destino não exista. No entanto, se existir, o usuário receberá uma mensagem de operação não permitida e o conteúdo do arquivo não será modificado. Vale ressaltar que, visando à simplificação no gerenciamento dos arquivos por parte dos usuários, o papel de administrador do sistema de arquivos foi removido do WFS. Assim, operações de superusuário, de modificação de permissões e de dono dos arquivos também foram desabilitadas.

A Figura 3.2 tem o objetivo de ilustrar, de forma simplificada, o comportamento de uma operação de cópia entre dois arquivos localizados dentro da mesma partição WORM. Para cada arquivo (ou diretório) envolvido nas operações do WFS, o FUSE executa a função *getattr* implementada no sistema de arquivos proposto. Assim, no exemplo apresentado, a função *getattr* será posta em execução duas vezes: uma contendo no argumento o PATH_A e outra com o PATH_B. Essas execuções ocorrem sempre antes da chamada à função principal e tem o objetivo de efetuar checagens relativas ao conteúdo e às permissões dos arquivos envolvidos na operação. Durante a execução da função de *getattr*, o WFS aplica algumas restrições WORM aos arquivos. Assim, se o arquivo referenciado pelo PATH_B já existir, um erro de falta permissão é gerado ao tentar modificar seu conteúdo, cancelando automaticamente a operação.

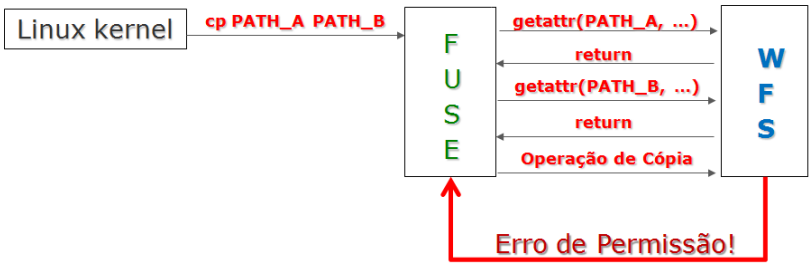

Figura 3.2: Exemplo de uma operação de cópia no WFS.

O diagrama das operações, quando executadas ao WFS, está exemplificado na Figura 3.3. Primeiramente, o usuário solicita a operação ao sistema operacional, por exemplo, criar uma cópia de segurança de um arquivo. Depois disso, o *kernel* do Linux identifica

que o arquivo a ser copiado encontra-se em uma partição gerenciada por um sistema de arquivos do FUSE, repassando a tarefa. Ao constatar que o WFS é o sistema de arquivos responsável pelo arquivo a ser duplicado, o FUSE, então, põe em execução as *syscalls* do WFS necessárias para completar a operação.

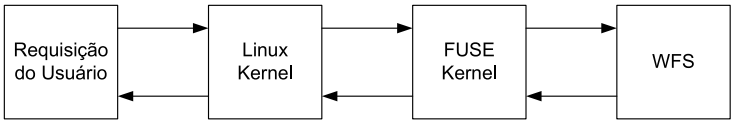

Figura 3.3: Diagrama de execução do WFS-FUSE

Embora a utilização do WFS reduza consideravelmente o risco de remoção e de modificação equivocada dos dados, é importante destacar que esse sistema não elimina a necessidade de realizar *backups* em dispositivos de armazenamento auxiliares (discos ópticos ou HD's externos), visto que esses componentes podem apresentar falhas que levem à perda de dados.

Para assegurar a proteção WORM fornecida pelo WFS, é importante que todas as operações sobre os dados que devem ser protegidos sejam executadas sobre a pasta gerenciada pelo WFS (o diretório-WORM), evitando o acesso ao diretório original, onde modificações de conteúdo são permitidas. Vale ressaltar que é aconselhável ocultar o diretório desprotegido.

O FUSE propicia benefícios como portabilidade entre sistemas operacionais, facilidade de desenvolvimento e possibilidade de utilização por parte de usuários sem privilégios de administrador. Em contrapartida, os sistemas baseados no FUSE apresentam uma perda de desempenho em relação aos sistemas de arquivos nativos do sistema operacional (como o Ext3). Assim, um estudo de avaliação de desempenho fez-se necessário para estimar a perda de desempenho apresentada pelo WFS em relação aos sistemas nativos não-WORM, como apresentado no Capítulo 4.

3.3 WFS: IMPLEMENTAÇÃO E COMPORTAMENTO

O FUSE fornece uma API de alto-nível, utilizando a linguagem C, com 26 funções que devem ser implementadas pelos desenvolvedores dos sistemas de arquivos virtuais [Sze09, Sin10b, MS01]. A partir dessa API, o WFS foi desenvolvido. Esta seção destina-se a fornecer os detalhes da implementação necessários para entender o funcionamento do sistema proposto. A Seção 3.3.1 apresenta o mecanismo de *trace* integrado ao WFS e a Seção 3.3.2 relaciona a API definida pelo FUSE ao comportamento do WFS.

3.3.1 Mecanismo de Rastreamento e de Depuração

Com o objetivo de compreender melhor a interação entre o FUSE e o WFS, além de prover aos desenvolvedores uma ferramenta eficiente e de utilização simples para depuração de código-fonte, um mecanismo de rastreamento foi incorporado ao sistema de arquivos

WORM. Esta funcionalidade permite aos desenvolvedores acompanhar, de forma clara, as informações de execução do WFS, sejam funções executadas, sejam argumentos recebidos ou valores processados.

Esse mecanismo gera, durante a execução do WFS, um arquivo com as informações solicitadas pelos desenvolvedores. Para tal, é necessário que seja habilitada a opção de modo de depuração no código-fonte do WFS, conforme a Figura 3.4. Ao atribuir o valor 1 à constante *DEBUG_MODE*, o rastreamento é ativado e um arquivo *log.txt* é criado com as informações da execução.

```
35
36 #define DEBUG_MODE 0
37
```

Figura 3.4: Trecho do código-fonte do WFS onde o mecanismo de rastreamento é habilitado.

A Figura 3.5 apresenta o código-fonte para inserção de informações relativas ao rastreamento ou à depuração do código do WFS. Uma vez que o desenvolvedor define os textos e os argumentos que devem ser inseridos, o arquivo gerado conterá todas as informações referentes à execução do WFS que foram solicitadas, seguindo a ordem cronológica para cada ocorrência. Vale ressaltar que o código das funcionalidades de depuração só será incorporado ao arquivo binário do WFS se o usuário habilitar a opção de rastreamento.

```
169 #if DEBUG_MODE
170         fprintf(log,"mkdir_%s\n",trpath);
171 #endif
```

Figura 3.5: Exemplo do trecho do código-fonte no qual dados de rastreamento ou de depuração são inseridos.

Este mecanismo foi fundamental na identificação de gargalos nas operações de leitura e de escrita de uma versão preliminar do WFS, uma vez que, a partir dele, foi possível obter a quantidade de *bytes* escritos por execução de funções como *callback_read* e *callback_write*. Para demonstrar outras possibilidade de utilização, este mecanismo de *trace* foi habilitado ao executar atividades corriqueiras com o WFS, tais como a criação de pastas e de arquivos, a leitura e escrita de conteúdo etc. A partir dos resultados obtidos, os diagramas de fluxo das operações implementadas pelo WFS e postas a executar pelo FUSE puderam ser definidos. A Seção a seguir apresenta tais diagramas.

3.3.1.1 Diagrama de Fluxo de Operações Antes de entrar nos detalhes de implementação da API pelo WFS, é fundamental que o leitor esteja ambientado com as composições de operações de baixo-nível que permitem ao WFS efetuar leituras, escritas,

cópias, entre outras atividades, envolvendo o conteúdo gerenciado pelo sistema de arquivos WORM. Para realizar este estudo, o WFS foi compilado com a opção de rastreamento ativada, assim, um arquivo de *log* foi gerado pelo sistema desenvolvido. A partir deste arquivo de rastreamento, um diagrama simplificado[2] das chamadas às funções do WFS foi elaborado. Assim, as requisições corriqueiras utilizadas por usuários residenciais de um sistema de arquivo foram rastreadas.

O processo de montagem envolveu dois diretórios: */home/tiago/Documentos/Normal/* e */home/tiago/Documentos/WFS/*. O primeiro consistia no diretório onde os dados estavam armazenados sem a proteção WORM, já o segundo foi o gerenciado pelo WFS. Um arquivo de cerca de 5 MB foi adicionado à pasta não-WORM antes da execução do estudo. Os resultados obtidos exibem detalhadamente o comportamento do WFS, evidenciando em quais situações as funções da API definidas pelo FUSE são executadas. Vale ressaltar que, para as situações suportadas pelo WFS, as operações no diretório WORM são realizadas diretamente nos arquivos da pasta não-WORM.

Logo após o processo de montagem do WFS, o FUSE executa as funções equivalentes à execução do comando *ls* do Linux para percorrer todo o diretório não-WORM. A partir deste momento, o FUSE é o responsável por colocar em execução as funções implementadas pelo WFS para solucionar as operações realizadas no diretório gerenciado pelo WFS. Para tal, o FUSE diferencia as ações em dois tipos: as realizadas sobre arquivos e sobre diretórios. Assim, funções diferentes são chamadas dependendo de qual operação foi solicitada e se essa manipulará arquivos ou diretórios. Assim, o WFS não é o responsável por atividades de baixo-nível, tais quais a verificação da ocorrência de ações dentro de seu diretório, nem tão pouco pela identificação de qual função deve ser executada. Portanto, a implementação do WFS pôde se ater a determinar o comportamento do sistema de arquivos, aplicando as características WORM à pasta destino.

Durante os testes do mecanismo de rastreamento no WFS foi possível constatar que, logo ao acessar pela primeira vez um diretório - independentemente da utilização do comando *cd* ou escrevendo em uma pasta diferente da acessada no momento da solicitação - o FUSE põe em execução a função *callback_access* para verificar as permissões de acesso à tal pasta. Contudo, como nesta solicitação caminhos de arquivos ou diretórios são fornecidos como argumentos ao FUSE, a função *callback_getattr*[3] é executada anteriormente à *callback_access*. A Figura 3.6 demonstra um diagrama simplificado das funções do WFS executadas pelo FUSE para acessar pela primeira vez um diretório.

Em distribuições baseadas em Linux, o comando *ls* é responsável por listar os arquivos pertencentes a um diretório. Como exposto na Figura 3.7, o comando *ls* tem o objetivo de listar os arquivos e as pastas localizados internamente ao diretório onde a busca foi solicitada. Para tal, a operação de leitura de diretório (*callback_readdir*) será executada,

[2]Vale esclarecer que, em algumas situações, o FUSE invoca operações como *callback_getattr* mais de uma vez para o mesmo diretório. Para efeito de simplificação, com o objetivo de facilitar o entendimento, algumas dessas ocorrências foram omitidas dos diagramas.

[3]A função *getattr* é a responsável pela verificação das propriedades dos arquivos gerenciados pelo WFS.

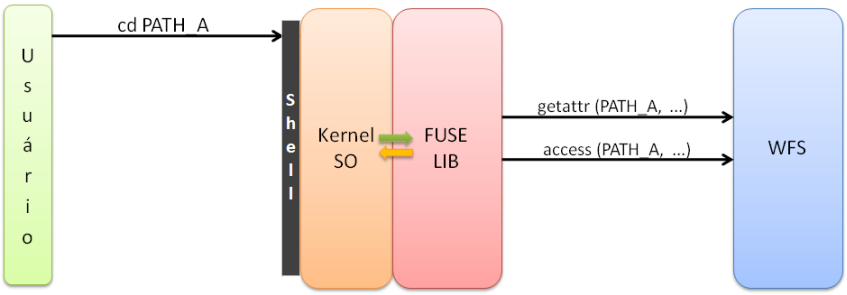

Figura 3.6: Diagrama de fluxo de operações para acesso de diretório.

associadas às chamadas à *callback_getattr* para todos os arquivos e os diretórios armazenados diretamente nesta pasta. Já o comando *ls -l*, além das execuções *callback_getattr*, realiza chamadas às funções *callback_getxattr* para cada arquivo de dentro do diretório verificado, obtendo, assim, as informações adicionais dos arquivos, como permissões, donos e datas (ver Figura 3.8).

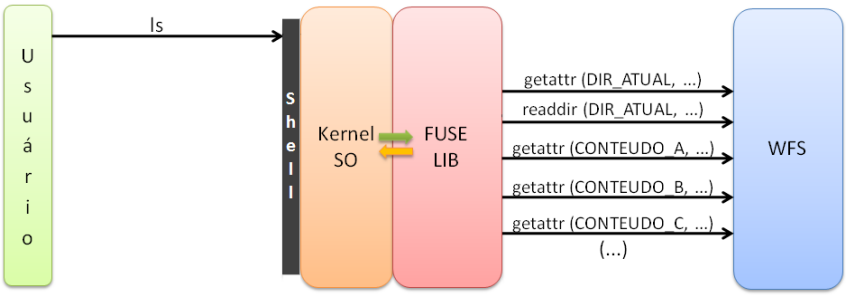

Figura 3.7: Diagrama das operações executadas pelo WFS para listar o conteúdo de um diretório.

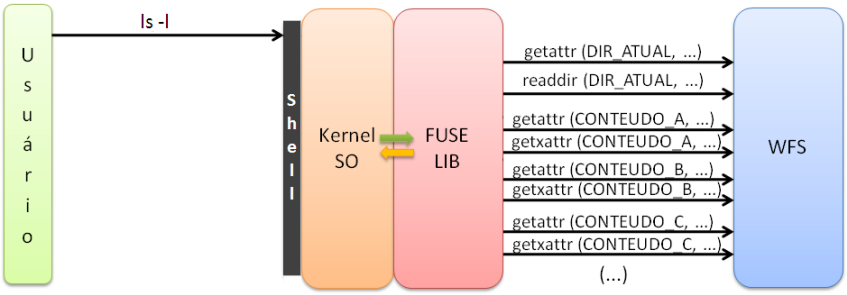

Figura 3.8: Diagrama para a chamada ao sistema "ls -l".

Os usuários utilizam o comando *mkdir* para criar um diretório dentro do caminho especificado. Esta solicitação dentro da pasta gerenciada pelo WFS envolve operações de *callback_getattr* sobre o diretório envolvido, além da função *callback_mkdir*, como pode ser observado na Figura 3.9. Para exemplificar a operação de criação de conteúdo, o comando *echo* foi utilizado, conforme apresentado na Figura 3.10. Este comando escreve

a frase passada como argumento em um arquivo também informado. Como requisições em arquivos necessitam de escrita de conteúdo nos mesmos, algumas diferenças sobre o processo de criação de diretórios podem ser percebidas: (i) a função *callback_create* será responsável pela criação de conteúdo; (ii) as operações de escrita de dados no arquivo recém-criado serão executadas (*callback_write*); (iii) por fim, o arquivo será fechado após a chamada à *callback_release*. Vale reforçar que o WFS não permite modificação de conteúdo, assim, caso o arquivo passado por argumento já exista, uma mensagem de erro de falta de permissão será apresentada pelo sistema operacional.

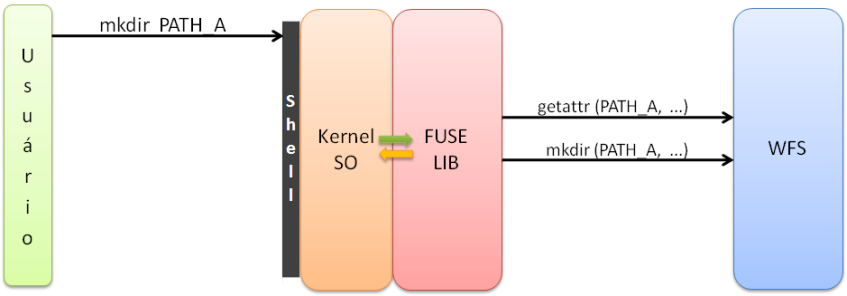

Figura 3.9: Diagrama para a criação de um diretório.

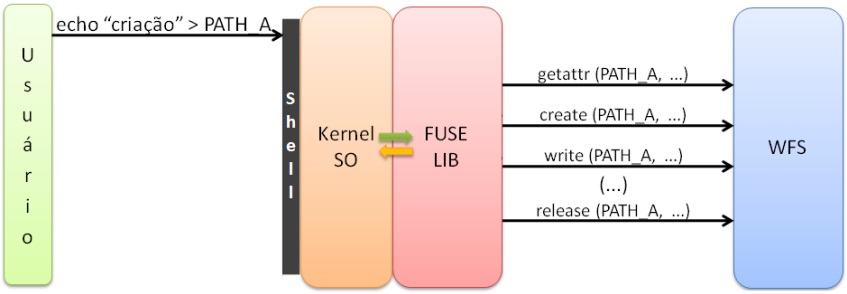

Figura 3.10: Diagrama para a criação de um arquivo.

O processo de cópia, comando *cp*, consiste na cópia integral de um conteúdo apontado pelo caminho-origem para um arquivo ou diretório apontado pelo caminho-destino. Para que esta operação seja permitida pelo WFS, o arquivo que receberá o conteúdo do arquivo original não pode existir previamente, sendo então criado pelo WFS durante o desenrolar do processo (ver Figura 3.11). As chamadas ao *callback_getattr* nos caminhos envolvidos são sucedidas por operações de abertura (*callback_open*) do arquivo origem e de criação do arquivo destino (*callback_create*). Vale ressaltar que após criar o arquivo, o mesmo é aberto em modo escrita. Assim, com os dois arquivos abertos, operações intercaladas de leitura (*callback_read*) no arquivo-origem e de escrita(*callback_write*) no arquivo-destino são executadas até que o conteúdo seja copiado integralmente. Por fim, os arquivos são fechados após a execução da função *callback_release*.

Assim como todas as requisições que envolvam arquivos e/ou diretórios, o processo de renomeação executa chamadas à *callback_getattr* tanto para os diretórios quanto para

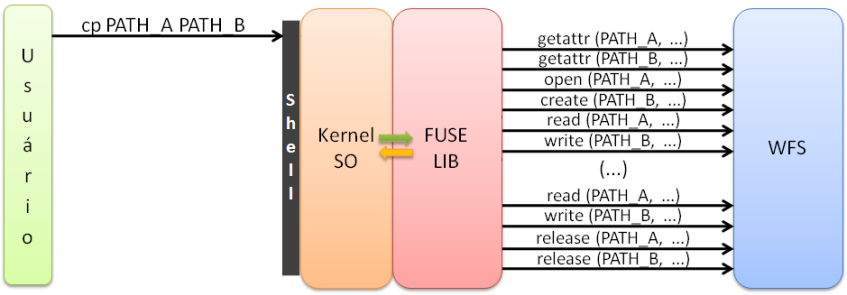

Figura 3.11: Diagrama para duplicar um arquivo ou diretório.

os arquivos envolvidos na operação. Como pode ser visto na Figura 3.12, após essa etapa, a função *callback_rename* será executada. Vale ser ressaltado que sistemas de arquivos Linux executam o mesmo conjunto de funções tanto para as operações de renomeação quanto de modificação de local de arquivos ou diretórios, assim, podem ser consideradas operações equivalentes. No entanto, como dispositivos WORMs não permitem modificação de conteúdo após a criação, um erro será gerado ao serem identificadas solicitações deste tipo.

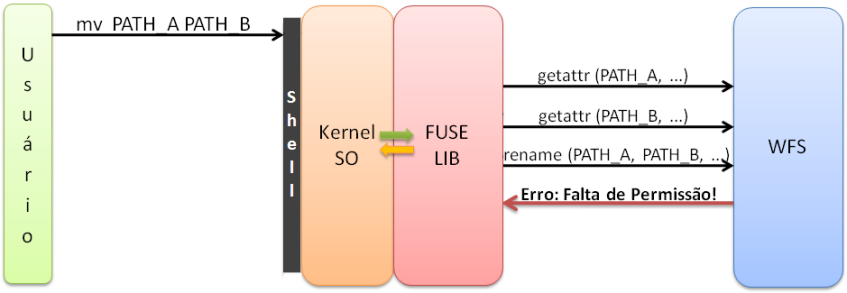

Figura 3.12: Diagrama para mover ou renomear um arquivo ou diretório.

O comando *chmod* é utilizado para solicitar modificações nas permissões dos arquivos em ambientes Linux. Por envolver arquivos, a *callback_getattr* é, mais uma vez, executada primeiramente, seguida pela função *callback_chmod* (ver Figura 3.13). Já para operações de substituição de dono, como apresentado na Figura 3.14, ficou evidenciado que as funções *callback_getattr* e *callback_chmod* são chamadas de forma similar ao processo de modificação de permissões de conteúdo. Adicionalmente, as funções *callback_getxattr* e *callback_chown* também são postas em execução.

Em sistemas Linux, usuários podem solicitar a remoção de um diretório através do comando *rmdir*, enquanto a operação equivalente sobre arquivos utiliza o *unlink*. Embora sistemas WORM não permitam que operações de remoção de conteúdo sejam realizadas, o diagrama para cada uma dessas situações foi gerado, como podem ser observados nas Figuras 3.15 e 3.16. O processo de remoção de diretórios envolvem chamadas à *callback_getattr* e à função *callback_rmdir*. No entanto, quando uma solicitação de remoção é efetuada sobre arquivos, a segunda função é substituída pela *callback_unlink*. Devido

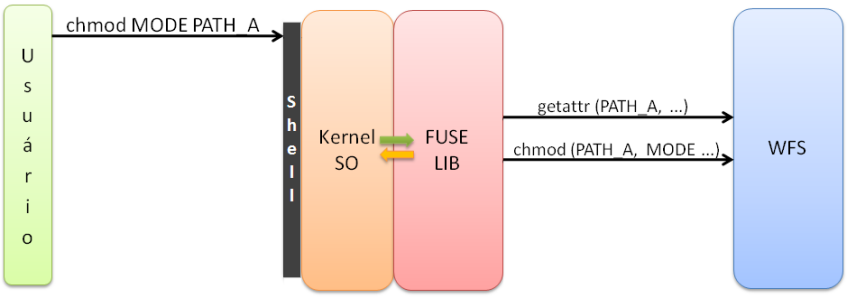

Figura 3.13: Diagrama para a operação de modificação de permissões.

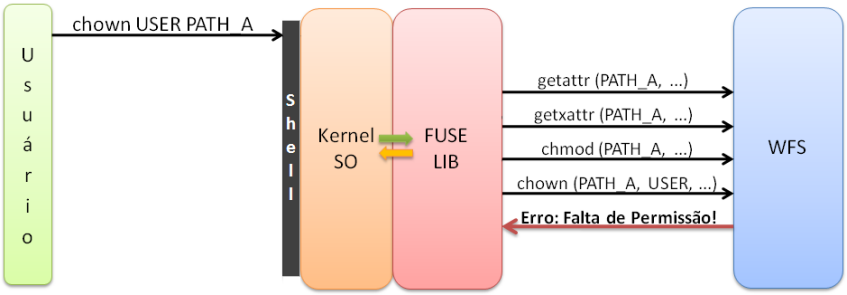

Figura 3.14: Diagrama para a operação de substituição de donos.

às restrições WORM, mensagem de erro de falta de permissão são exibidas para tais solicitações.

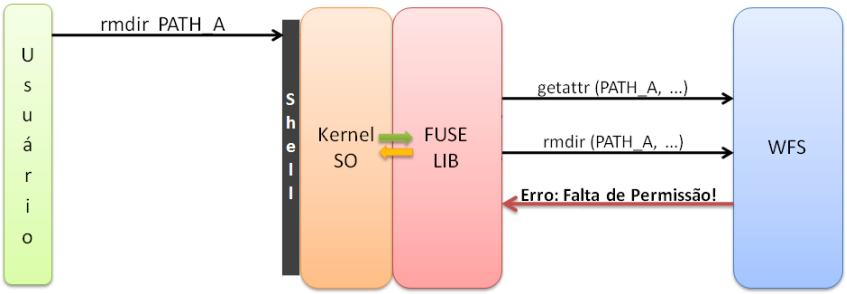

Figura 3.15: Diagrama das operações executadas pelo WFS para apagar um diretório.

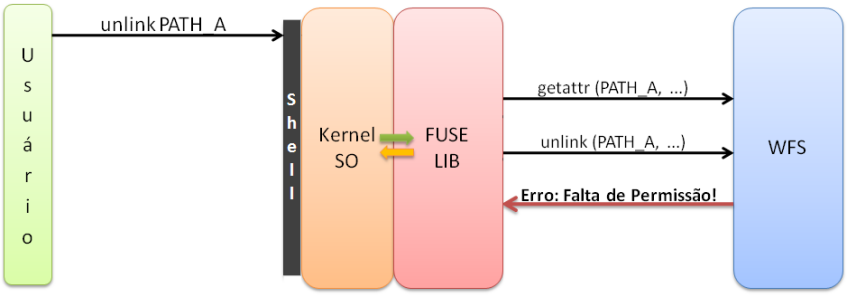

Figura 3.16: Diagrama para a remover um arquivo.

Uma operação de leitura de conteúdo envolve, além da chamada à função *getattr* - passando o dado a ser lido como argumento -, as execuções das funções responsáveis por abrir o arquivo/diretório em modo leitura e por ler o conteúdo solicitado, respectivamente. Por fim, o arquivo é fechado. Vale reforçar que o FUSE diferencia funções que manipulam arquivos das que trabalham com diretórios. Como exemplificação desta operação, um arquivo de áudio foi posto em execução utilizando o *software* Rhythmbox, como pode ser observado na Figura 3.17. Foi possível notar as chamadas à *callback_getattr*, seguida pela abertura e leitura de parte do arquivo, funções *callback_open* e *callback_read*, respectivamente. Logo após, o arquivo foi fechado utilizando a função *callback_release*. Vale ressaltar que a execução dessas três últimas funções se repetiu diversas vezes ao longo da exibição do arquivo multimídia, visto que o mesmo não foi carregado integralmente na primeira ocorrência.

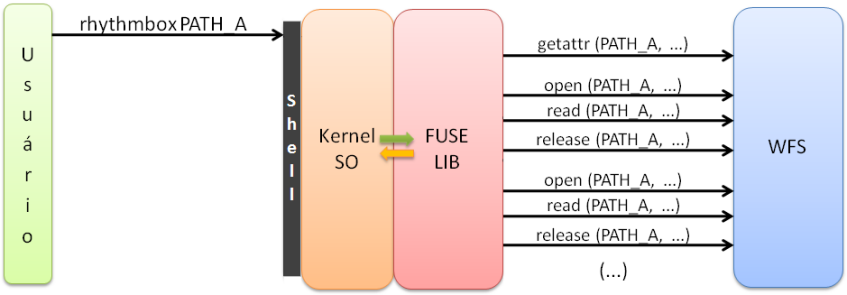

Figura 3.17: Diagrama para a operação de um arquivo utilizando o software multimídia Rhythmbox.

A operação de modificação de conteúdo de um arquivo se diferencia da de leitura a partir do momento no qual a abertura do arquivo ou do diretório é solicitada, visto que uma requisição de abrir em modo escrita será enviada pelo FUSE à função encarregada dessa atividade implementada pelo WFS. Em sistemas de arquivos normais, após esta solicitação, as funções responsáveis pela escrita de conteúdo no arquivo seriam postas em execução. No entanto, o WFS bloqueia solicitações de abertura de arquivo em modo escrita a partir da função *callback_open*, lançando um erro de falta de permissão. Assim, apenas os arquivos recém-criados podem ser escritos, visto que são abertos pela função *callback_create*.

A partir dos resultados obtidos ao utilizar o mecanismo de rastreamento implementado pelo WFS, foi possível constatar quais funções do WFS são chamadas quando *syscalls* de ambiente Linux são executadas por usuários. Assim, o comportamento do FUSE/WFS pode ser compreendido mais detalhadamente para operações encarregadas de apagar, de renomear, de ler, de escrever, de modificar as propriedades dos arquivos etc. Ficou evidente a importância da função *getattr* que é executada pelo FUSE para cada arquivo ou diretório localizado dentro da pasta WORM envolvido na operação. Devido à essa característica, algumas das restrições WORM foram atribuídas aos arquivos dentro desta função. Em virtude do excesso de chamadas à *getattr*, é fundamental garantir uma execução rápida dessas ações para evitar perda de desempenho.

3.3.2 Funções da API

O código-fonte do WFS implementa toda a API de alto-nível definida pelo FUSE. Após o WFS ser montado, será possível visualizar no diretório WORM os mesmos arquivos presentes na pasta desprotegida. Caso o usuário opte por realizar ações dentro do diretório não-WORM, o SO executará chamadas ao sistema nativo (o Ext3, por exemplo) para resolver essa solicitação. Contudo, se a ação ocorrer no diretório gerenciado pelo WFS, o FUSE será informado que deverá solucionar a requisição. Após descobrir quais operações devem ser executadas, o FUSE, então, executa as funções do WFS que resolvem a solicitação do usuário, seguindo as operações descritas na Seção 3.3.1.

Para que o *Filesystem in Userspace* esteja habilitado a realizar chamadas às funções do WFS, a estrutura *fuse_operations* deve ser definida dentro do código-fonte do sistema WORM, como apresentado na Figura 3.18. Nela, cada uma das funções do WFS é associada ao apontador equivalente à função da API do FUSE a qual representa.

Vale ressaltar que, por se tratar de um sistema de arquivos virtual, algumas funções implementadas apenas repassam chamadas ao sistema, tornando o código mais conciso do que os sistemas de arquivos nativos. Assim, durante a implementação, foi possível concentrar quase a totalidade dos esforços nas funcionalidades WORM que seriam fornecidas ao usuário, visto que o FUSE se encarrega de realizar as operações de baixo-nível.

A função *callback_getattr* é a responsável pela obtenção dos atributos dos arquivos e dos diretórios gerenciados pelo WFS. Algumas das funcionalidades WORM fornecidas pelo WFS foram implementadas nessa função. Essa escolha é justificada pelo comportamento do FUSE, que sempre executa a *getattr* antes de realizar operações com arquivos, tais como criar, remover, modificar ou mover diretórios ou arquivos. O FUSE chama a *getattr* passando, como parâmetro, cada um dos caminhos de arquivos (*paths*) envolvidos. Dessa forma, operações de remoção de arquivos, de modificação de conteúdo e de superusuários podem ser bloqueadas para os arquivos dentro da partição WORM. A Figura 3.19 mostra o trecho de código responsável por impedir essas operações, desabilitando as permissões de escrita em todos os arquivos do diretório-WORM.

A função *callback_mkdir* é executada quando há uma solicitação de criação de diretório dentro da partição controlada pelo WFS. Caso o diretório apontado pelo argumento *path* não exista, ele será criado. Depois de criados, a *callback_readdir* é a função da API do FUSE encarregada de efetuar as leituras dos diretórios WORM.

A *callback_rmdir* é utilizada pelo FUSE para solicitar a remoção de diretórios gerenciados por sistemas de arquivos nele baseados. No entanto, a política WORM impede que os diretórios sejam removidos, portanto, essa função retorna um erro de falta de permissão ao ser executada.

A *callback_create* é a função encarregada de criar o arquivo se o mesmo não existir. Após ser criado, o arquivo é aberto em modo escrita e, logo em seguida, todos o seu conteúdo é armazenado definitivamente na pasta WORM. Após esse momento, o arquivo

```
459 struct fuse_operations callback_oper = {
460     .mknod      = callback_mknod,
461     .symlink    = callback_symlink,
462     .unlink     = callback_unlink,
463     .rmdir      = callback_rmdir,
464     .rename     = callback_rename,
465     .link       = callback_link,
466     .chmod      = callback_chmod,
467     .chown      = callback_chown,
468     .utime      = callback_utime,
469     .getattr        = callback_getattr,
470     .readlink       = callback_readlink,
471     .readdir        = callback_readdir,
472     .mkdir      = callback_mkdir,
473     .truncate   = callback_truncate,
474     .open           = callback_open,
475     .read           = callback_read,
476     .write      = callback_write,
477     .statfs         = callback_statfs,
478     .release    = callback_release,
479     .fsync      = callback_fsync,
480     .access     = callback_access,
481     .create     = callback_create,
482     .setxattr   = callback_setxattr,
483     .getxattr   = callback_getxattr,
484     .listxattr  = callback_listxattr,
485     .removexattr= callback_removexattr
486
487 };
```

Figura 3.18: Trecho do código-fonte do WFS onde é feita a associação entre as funções implementadas no WFS e a API provida pelo FUSE.

não poderá mais ser modificado. Uma vez criado, o arquivo não poderá ser sobrescrito a partir do WFS.

Encarregada pelo FUSE de abrir um arquivo dentro de uma partição WORM, a *callback_open* permite sem ressalvas a abertura de um arquivo em modo leitura. Caso os flags de abertura indiquem que o arquivo deve ser modificado ou removido, essa operação é proibida e um erro de falta de permissão é gerado. Nessa função o desempenho do WFS pode ser ajustado através da configuração dos parâmetros *Direct I/O* e *Keep Cache*[4] (ver Figura 3.20). A Figura 3.21 apresenta o trecho do código-fonte do WFS responsável por impedir a abertura de arquivos no modo de apenas-escrita (flag O_WRONLY), de leitura-escrita (O_RDWR) e de remoção (O_EXCL). Assim, os arquivos não poderão ser modificados ou apagados após sua criação.

Para realizar as operações de leitura e de escrita em um arquivo é necessário que este seja aberto previamente pelas funções *callback_create* ou *callback_open*. A função *callback_read* é destinada à leitura de informações de um arquivo. O WFS impede, através de outras funções, que haja a modificação nos arquivos salvos anteriormente. Assim, a

[4]A configuração dos parâmetros de configuração do WFS será abordada no capítulo de avaliação de desempenho, pois impactam diretamente no desempenho do sistema desenvolvido.

```
83      // Remove write permissions => chmod a-w
84      st_data->st_mode &= ~(S_IWUSR | S_IWGRP | S_IWOTH);
85      return 0;
```

Figura 3.19: Trecho da função getattr no qual as modificações nos arquivos já criados são proibidas.

```
333
334     finfo->keep_cache = 1;
335     finfo->direct_io = 1;
336
```

Figura 3.20: Trecho da função open no qual parâmetros de configuração do WFS podem ser ajustados.

operação de leitura dos dados estará sempre habilitada para todos os arquivos. Já a *callback_write* é encarregada da escrita de conteúdo em um arquivo já aberto. Essa operação só poderá ser realizada para arquivos recém-criados através da *callback_create*, uma vez que o *callback_open* impede a modificação de arquivos já existentes.

Função do FUSE destinada à remoção de arquivos, a *callback_unlink* retorna um erro de falta de permissão ao ser executada, visto que, em dispositivos WORM, depois de ser criado, o conteúdo não podem ser excluído. Além de não permitir a remoção, os dispositivos *Write-Once Read-Many* desabilitam também a modificação do nome dos arquivos. Logo, a função *callback_rename*, responsável por essa tarefa, retornará um erro de falta de permissão ao ser requisitada pelo FUSE.

A *callback_truncate* altera o tamanho do arquivo para o exato valor passado como argumento durante a chamada da função. O FUSE só executa essa operação logo após o arquivo ser criado ou para arquivos em processo de modificação. Se o arquivo for maior que o tamanho solicitado, os dados extras serão perdidos. Já, se o arquivo for menor, ele será estendido e a parte acrescentada receberá *bytes* nulos (‘\0’). O offset do arquivo não será modificado.

A *callback_chmod* é encarregada de alterar os bits de permissão dos arquivos de dentro da partição WORM. Uma vez que o WFS pretende facilitar o gerenciamento dos arquivos, o papel do administrador do sistema de arquivos foi eliminado. Assim, operações de superusuário, modificações de permissões e donos do arquivo foram desabilitadas. Como a operação de modificação de permissão de arquivos é também utilizada pelo sistema operacional, erros de falta de permissão não puderam ser lançados. Assim, por definição em tempo de projeto, as permissões do arquivo serão mantidas com um valor constante, independentemente da execução dessa operação.

No que se refere à operação de modificação de proprietário ou de grupo ao qual

```
307     int flags = finfo->flags;
308
309     if ((flags & O_WRONLY) ||
310           (flags & O_RDWR) ||
311            (flags & O_EXCL)){
312         return -EPERM;
313     }
```

Figura 3.21: Trecho da função open do WFS no qual um erro de falta de permissão é gerado ao solicitar modificações e ou remoções de arquivos já existentes.

pertence determinado arquivo ou diretório, o comando Linux *chown* é executado pelos administradores do sistema. No entanto, seguindo o requisito que busca simplificar o gerenciamento de arquivos do WFS (ver Seção 3.1.1), a *callback_chown* implementada pelo sistema proposto gera um erro de falta de permissão ao executar.

A função *callback_mknod* seria a responsável por criar um *file node* nos diretórios controlados pelo WFS. Contudo, seguindo o requisito que visa simplificar o gerenciamento de arquivos, essa função retorna um erro de falta de permissão, visto que os arquivos podem ser criados pela função *callback_create* e os diretórios pela *callback_mkdir*.

A *callback_utime*, em sistemas de arquivos convencionais, é a responsável por atualizar a hora de acesso e de modificação do arquivo. Como no WFS os arquivos não podem ser modificados, essa função também retorna um erro de falta de permissão.

A *callback_access* é a responsável pela verificação das permissões de acesso aos arquivos e pastas. Devido a isso, algumas restrições da política *Write-Once Read-Many* podem ser aplicadas durante sua execução, como, por exemplo, a proibição de modificação dos arquivos, permitindo apenas as operações de leitura dentro do diretório WORM. A Figura 3.22 traz o trecho de código onde as modificações de conteúdo são bloqueadas. O flag W_OK sinaliza quando é solicitado o acesso a um arquivo em modo escrita.

```
464
465     if (mode & W_OK) return -1;
466
```

Figura 3.22: Trecho do código-fonte da função access onde é bloqueado o acesso a um arquivo em modo de modificação.

A *callback_symlink* é a função encarregada da criação de um *link* simbólico para um arquivo real. Os *Links* simbólicos ou *soft links* são arquivos que apresentam como conteúdo o caminho para um arquivo real. Esses *links* são, na verdade, apontadores, caso o arquivo original seja apagado, tornam-se *links* mortos. Para essa versão do WFS, visando facilitar o gerenciamento dos arquivos, a *callback_symlink* foi desabilitada. Apesar da criação de

links simbólicos estar proibida nessa versão do WFS, os *soft links* que, por ventura, já existam nos dados do usuário podem ser lidos através da função *callback_readlink*, que se encontra habilitada.

Em sistemas Linux, o *Hard link* é um apontador para o *inode* do arquivo alvo. Arquivos ligados dessa forma são exatamente iguais. Caso um deles seja removido, o outro continuará com o mesmo conteúdo. A função *callback_link* seria responsável por criar um *hard link* entre arquivos. No entanto, de forma equivalente aos *links* simbólicos, essa funcionalidade foi desabilitada. Vale ressaltar que o sistema operacional destina aos *hard link* as mesmas funções de abertura, de leitura e de escrita dos arquivos convencionais, uma vez que esses *links* não possuem diferenças para os arquivos tradicionais.

A *callback_fsync* é responsável pela sincronização do conteúdo de um arquivo, escrevendo imediatamente os dados contidos no *buffer* no dispositivo de armazenamento. No WFS, essa função não possui implementação relevante. Já a *callback_statfs* retorna informações sobre o *filesystem* no qual o arquivo-alvo está armazenado.

Os *extended attributes* são pares formados por (nome;valor) que permitem aos usuários associar aos retardados do arquivo informações que não são interpretadas pelo sistema de arquivo [Lin09]. A *callback_setxattr* está associada a função da API do FUSE responsável por alterar o valor do *extended attribute*. No entanto, visando manter o padrão de implementação, funções de modificação de conteúdo geram um erro de falta de permissão. Assim, a função *callback_removexattr*, responsável por remover um *extended attribute* existente, também retorna um erro de falta de permissão. A *callback_getxattr* retorna o valor armazenado em um *extended attribute*. Já a função *callback_listxattr* é encarregada de listar os nomes dos *extended attribute* existentes.

A função *callback_release* libera o espaço utilizado por um arquivo ainda aberto quando não existem *descriptors* abertos e nem tão pouco mapeamento do conteúdo em memória. O único diferencial da implementação do WFS para o padrão do FUSE é o reforço na proibição de modificação do conteúdo.

3.4 CONSIDERAÇÕES FINAIS

Este capítulo discorreu sobre o WFS, um sistema de arquivos baseado na infraestrutura FUSE que tem por objetivo fornecer aos usuários Linux residenciais maior proteção contra remoções e alterações involuntárias de arquivos. Este sistema apresenta funcionalidades baseadas nos sistemas *Write-Once Read-Many*, tais como a proibição de alteração e de remoção de conteúdo dentro da partição protegida, embora a criação e a leitura de conteúdo sejam liberadas. Além disso, este capítulo apresentou outros requisitos do sistema também atendidos pelo WFS, como: ser código-livre; atender a usuários sem privilégios de administração; ser instalado e configurado facilmente; possibilitar uma forma alternativa de modificação e de remoção de dados, além de permitir o gerenciamento dos dados de forma simples. Com o objetivo de assegurar que o sistema desenvolvido supra todos os requisitos elicitados a partir das reuniões semanais com o cliente, um estudo de avaliação de desempenho foi realizado, conforme será discutido no Capítulo 4.

CAPÍTULO 4

AVALIAÇÃO DE DESEMPENHO

Um dos requisitos do *Write-Once Read-Many File System* é apresentar desempenho similar ao Ext3 não-WORM, uma vez que, por ser um sistema de arquivos no espaço do usuário, é esperado que o WFS apresente menor desempenho em relação aos sistemas nativos. Assim, foi realizada uma avaliação de desempenho com o objetivo de demonstrar que a diferença de desempenho entre o WFS e o Ext3, para determinada carga de trabalho, é suficientemente pequena (menor que 20%), ou seja não deve ser suficiente para dificultar a utilização do sistema WORM por parte de usuários residenciais. Este capítulo discorre sobre esta avaliação. Para tal, a carga de trabalho *Bonnie* [Bra10] foi adaptada para o contexto WORM e aplicada no estudo. Neste capítulo são apresentadas a metodologia de avaliação, assim como a definição dos experimentos e da carga de trabalho utilizada. Por fim, os resultados da avaliação de desempenho são discutidos.

4.1 METODOLOGIA ADOTADA

O processo de avaliação de desempenho aplicado no WFS consistiu de 5 grandes fases, conforme disposto no diagrama apresentado na Figura 4.1. Este processo foi aplicado em todas as cargas de teste, sempre explorando todos os tamanhos de arquivos utilizados. Esta seção tem por objetivo discorrer sobre cada uma dessas etapas, porém sem entrar em detalhes sobre os resultados específicos dos cenários de teste. Os resultados obtidos para as cargas de trabalho são detalhados nas Seções 4.4 e 4.5.

Figura 4.1: Diagrama do processo da avaliação de desempenho aplicado ao WFS

A primeira fase do processo de avaliação de desempenho é a etapa de configuração do ambiente na qual serão realizadas as medições. Esta fase tem por objetivo assegurar que o sistema esteja livre de interferências (ruídos) significantes, ou seja, que possam influenciar nos resultados das medições. Para atingir esta meta, o sistema deve ser cuidadosamente analisado e todos os processos que não sejam necessários para execução dos experimentos

devem ser desativados. Também vale ressaltar que, durante o processo de execução dos experimentos, o sistema deve ser mantido fora de qualquer rede. Após essa fase, o ambiente deve ser submetido a uma avaliação preliminar. É fundamental que, para todas as métricas de desempenho consideradas no estudo, os resultados obtidos com sistema sem carga apresentem amplitude insignificante em relação aos obtidos ao executar o WFS utilizando a carga de trabalho definida. Dessa forma, para todos os cenários de teste, o ambiente pode ser considerado controlado.

A segunda etapa consiste na definição das métricas que serão importantes para avaliação de desempenho, bem como da carga de trabalho (*workload*) que será adotada. A escolha criteriosa da carga de trabalho é fundamental para assegurar que o sistema, de fato, seja avaliado de forma relevante, realizando comparações de comportamento do mesmo em situações significativas para o estudo[1]. Após a fase de especificação das métricas e dos *workloads*, as ferramentas de monitoração devem ser selecionadas ou adaptadas. O tempo de execução e a utilização de recursos como disco e processador são exemplos de métricas que podem ser monitoradas através destas ferramentas.

Após a definição das métricas e do *workload*, deve ser iniciada a fase de seleção das técnicas da área de Avaliação de Desempenho de Sistemas que serão empregadas no estudo. Técnicas de projeto de experimentos [Jai08] devem ser utilizadas quando é possível observar a existência de diferenças relevantes de desempenho, dependendo de como os parâmetros de configuração de um sistema são ajustados. Através do experimento fatorial é possível obter a relevância de cada fator de configuração no desempenho do sistema, e, assim, sugerir uma configuração que melhore o desempenho do sistema. Associada à utilização das técnicas de experimento fatorial, é possível realizar a comparação de desempenho entre sistemas distintos fornecendo um resultado com significância estatística. Ambos, experimentos fatoriais e comparação estatística entre sistemas distintos, são utilizados na avaliação de desempenho deste trabalho.

Na fase seguinte, os experimentos devem ser conduzidos com o ambiente devidamente controlado e utilizando a carga de trabalho selecionada. As métricas definidas devem, então, ser obtidas através das ferramentas de monitoração para todos os cenários de teste. É importante ressaltar que estes cenários devem ser testados em ambientes semelhantes, reduzindo o risco de interferências relevantes aos experimentos. Por fim, os resultados dos experimentos devem ser analisados. Os resultados da avaliação de desempenho deste trabalho são descritos na Seção 4.4.

4.2 CARGA DE TRABALHO

Neste trabalho, a avaliação de desempenho do WFS foi realizada utilizando uma adaptação do *Bonnie Workload Tool* [Bra10]. Apesar de adotar um padrão, o *Bonnie* permite que seja determinado o tamanho do arquivo com o qual os testes serão executados. Com o objetivo de testar o comportamento do WFS para arquivos de tamanhos diferentes, neste estudo foram utilizados arquivos de 10MB e de 1000MB. Esses tamanhos foram escolhi-

[1] A definição detalhada do *workload* utilizado neste estudo pode ser encontrada na Seção 4.2

dos arbitrariamente. Contudo, o objetivo principal foi avaliar o comportamento dos FS ao executar operações em arquivos de tamanhos semelhantes aos encontrados no formato *mp3* (na ordem de 10MB), bem como em arquivos de vídeo (na ordem de 1000MB).

O *benchmark Bonnie* avalia o desempenho de um sistema de arquivos através de diversos testes de leitura e de escrita, como pode ser observado na Tabela 4.1. Primeiramente, um arquivo do tamanho especificado pelo usuário é escrito utilizando a função *putc()*, ou seja, é escrito um caractere por vez. Na segunda etapa, o arquivo é escrito utilizando a função *write()*, em blocos de 16KB. Nas duas próximas fases, o arquivo é lido um caractere por vez, com *getc()* e em blocos de 16KB, com a função *read()*, respectivamente. Por fim, essas duas operações de leitura são executadas novamente, só que, desta vez, a *page cache* é esvaziada antes de cada execução. Além dos testes listados na Tabela 4.1, o *Bonnie* também mede o desempenho do sistema através de testes concorrentes de busca, utilizando funções *lseek* [RG10].

Tabela 4.1: Funções executadas nos testes de leitura e de escrita realizados pelo *Bonnie*

Tipo de teste	**Operações com Bytes**	**Operações com Blocos**
Escrita	putc	write
Leitura	getc	read

Devido às restrições intrínsecas WORM, como proibição de reescrita e de remoção de conteúdo, algumas operações do *Bonnie* tiveram de ser substituídas. O código-fonte desta adaptação encontra-se disponibilizado na Web de forma livre e gratuita [Fal10]. Vale ressaltar que sempre se buscou modificar minimamente o comportamento do *workload* original. Como principais alterações, podemos citar:

- o redirecionamento de operações de reescrita de conteúdo, que na versão adaptada está sendo redirecionada para outro arquivo. Assim, a característica WORM de proibição de modificação da informação não é infringida;
- o código responsável por apagar os arquivos criados pelo *Bonnie* para realizar os testes foi removido da versão adaptada. Na versão modificada por este trabalho, os arquivos criados pelo *Bonnie* continuam existindo mesmo após o encerramento da execução deste *benchmark*.

Esta carga de trabalho foi aplicada diretamente em partições controladas pelo WFS, bem como diretamente sobre o Ext3 sem características WORM. A descrição detalhada da realização dos experimentos é apresentada nas seções seguintes.

4.3 DEFINIÇÃO DOS EXPERIMENTOS

A avaliação de desempenho realizada neste trabalho investiga a existência de diferença relevante de desempenho entre o WFS e o Ext3, comparando o comportamento desses

sistemas em um ambiente real. Para tanto, um notebook Dell 1500 de processador Intel Core 2 Duo 2,20 GHz, 2,5 GB de RAM, 160 GB de HD IDE e Sistema Operacional Ubuntu 9.04 foi utilizado. O Apêndice B traz a descrição detalhada da configuração do *hardware* utilizado para a medição dos experimentos. A fim de evitar interferências e *outliers* no experimento, o sistema foi desconectado de qualquer rede e apenas os processos necessários estavam em execução [2]. Além disso, uma análise preliminar foi realizada no ambiente. Essa análise indicou a inexistência de interferências nas métricas presentes neste estudo, portanto o ambiente pôde ser considerado controlado.

O *Bonnie* é utilizado como carga de trabalho nas duas etapas da avaliação de desempenho do WFS descritas neste trabalho:

1. técnicas de experimentos fatoriais completos [Jai08] são utilizadas com o objetivo de descobrir ajustes de configuração que aumentem o desempenho do sistema de arquivos WORM, considerando algumas possibilidades de configuração como cenários de teste;

2. os cenários que apresentaram melhor desempenho são confrontados com os resultados obtidos, aplicando o *workload* diretamente no Ext3, sem a proteção WORM [Jai08, ET97, FM03].

Para tal estudo, diversas métricas foram consideradas. A principal métrica considerada neste estudo, o tempo de execução, foi obtida através da primitiva *time* [Chr94] do Linux. Ao ser executado, este comando retorna o tempo real (em segundos) transcorrido entre o início e o final da execução do *benchmark*. Além do tempo de execução, este estudo considerou as métricas fornecidas pelo próprio *workload Bonnie*. São elas: as taxas de leitura e de escrita de bytes e o desempenho em operações concorrentes de busca [Bra10].

O FUSE disponibiliza alguns parâmetros de configuração para que os desenvolvedores/usuários ajustem o comportamento do sistema de arquivos à necessidade do ambiente no qual ele se insere [Sze09, Sze05, Dut02, Eat07]. Vale ressaltar que a configuração desses atributos pode influenciar de forma significativa no desempenho do sistema. Embora possam existir outros fatores de configuração do FUSE, nesse trabalho apenas 4 deles foram encontrados. Esses fatores, definidos na Tabela 4.2, foram utilizados em um experimento fatorial completo, descrito a seguir. Vale salientar que cada fator pode assumir apenas dois níveis: ligado (1) e desligado (0). Com o objetivo de facilitar a visualização dos resultados, os fatores são referenciados neste trabalho por: A, B, C e D.

Os fatores apresentados na Tabela 4.2 podem ser subdivididos em dois grupos: Opção de montagem e de código-fonte. O primeiro grupo se refere aos atributos que podem ser habilitados ou desabilitados durante a operação de montagem do sistema de arquivos, sem a necessidade de realizar alterações no código-fonte. Já para habilitar os fatores de código-fonte, o usuário terá que modificar e compilar o código do sistema de arquivos.

[2] Os processos responsáveis pela atualização do SO, pelo gerenciamento de rede, pela execução de *softwares* aplicativos, entre outros, foram desabilitados

Tabela 4.2: Fatores de configuração do WFS

Rótulo	Nome do Fator	Detalhe
A	Direct I/O	Opção de Montagem
B	Kernel *cache*	Opção de Montagem
C	Keep *cache*	Código-Fonte
D	Direct I/O	Código-Fonte

Ao utilizar a opção de montagem *Direct I/O*, o módulo do FUSE localizado no kernel do Sistema Operacional desabilita a escrita na *page cache*, reduzindo o número de cópias adicionais na memória para um. Assim, as operações de escrita tornam-se mais rápidas, porém, o desempenho de leitura pode sofrer algum impacto negativo [RG10]. A opção *Kernel cache* desabilita a remoção dos dados presentes na *cache*. Essa opção deve ser habilitada apenas em sistemas de arquivos cujos dados nunca são modificados externamente, ou seja, não podem ser modificados por sistema de arquivos que não se utilizam do FUSE [Sze05]. Considerando os fatores de código-fonte, o *Keep cache* indica que o conteúdo dos arquivos armazenados em *cache* não precisa ser removido, já o fator de *Direct I/O*, habilita a utilização do *Direct I/O* nos arquivos [Sze09].

A execução de experimentos fatoriais completos é indicada em casos cujos resultados pretendidos são o efeito e a relevância de cada conjunto de fatores, considerando os diversos cenários de teste. Assim, para os quatro parâmetros de configuração selecionados do WFS, dezesseis casos de teste foram gerados. A técnica aplicada fornecerá recurso para indicar qual(is) o(s) ajuste(s) dos parâmetros responsável(eis) pelo menor o tempo de execução para a carga de trabalho utilizada. A partir dessa indicação, será possível detalhar quais dos fatores testados são, de fato, relevantes para melhorar o desempenho do sistema de arquivos WORM.

Para esse experimento, o *Bonnie* adaptado foi utilizado como carga. Uma partição de 50GB foi formatada com o sistema de arquivo nativo Linux Ext3. Esta partição possui dois diretórios: um deles gerenciado diretamente pelo Ext3, outro pelo WFS. Vale salientar que o WFS foi montado sobre um diretório Ext3 e que todos os 16 cenários possíveis foram testados em um experimento fatorial completo, considerando os 4 fatores apresentados na Tabela 4.2.

Foram realizadas 30[3] medições de cada um dos cenários de teste, tanto para arquivos de 10MB quanto para os de 1000MB. Vale destacar que uma avaliação com o mesmo processo de medição também foi executada diretamente no Ext3. Neste trabalho, os cenários de melhor desempenho do WFS são confrontados com os valores obtidos diretamente do Ext3, para os dois tamanhos de arquivo. Além das métricas originais do *Bonnie*, o tempo de execução foi medido. Um procedimento similar foi executado para a realização de medições diretamente no Ext3.

[3]Testes preliminares apontaram que a realização de 30 repetições para cada cenário de teste fornece o nível de confiança desejado para o processo de avaliação.

Para efetuar as comparações entre o WFS e o Ext3, diversos testes estatísticos foram executados para oferecer credibilidade à análise. Assim, testes de normalidade e testes do tipo t foram realizados com 95% de nível de confiança. Para todos os sistemas considerados, os intervalos de confiança e os intervalos interquartis são destacados.

4.4 TEMPO DE EXECUÇÃO

Após a fase de execução dos experimentos, a etapa de análise foi iniciada. No entanto, antes da apresentação dos resultados, é importante explicar brevemente a notação que será utilizada para referenciar as configurações dos fatores do WFS daqui por diante. Cada fator é representado por um dígito e referenciados pelos rótulos: A, B, C e D. Cada dígito pode assumir 3 valores: 0, 1, x. A configuração 0000, por ser a configuração onde todos os fatores estão desabilitados, será considerada, neste estudo, a configuração-padrão do WFS.

Quando o respectivo fator estiver habilitado, o seu dígito será sinalizado com o valor '1'. Nas situações em que os fatores estejam desligados, o valor '0' será atribuído. Em algumas situações, para evitar que grande quantidade de configurações sejam escritas em sequência, a letra 'x' designará que a referida característica ocorre independentemente do respectivo fator estar ativado ou não, ou seja, sendo equivalente ao sinal de *don't care* no estudo de circuitos digitais. Para exemplificar, a configuração 1xxx, se refere a todas as 8 configurações que possuem o fator A ligado (1000, 1001, ..., 1110, 1111), enquanto a 1000, refere-se apenas à configuração na qual somente o fator A está habilitado.

A adaptação do *workload Bonnie* foi utilizada considerando dois tamanhos de arquivos, 10MB e 1000MB. Esta seção tem por objetivo discorrer sobre os resultados dos experimentos fatoriais completos, obtidos a partir das medições realizadas, separando-os de acordo com o tamanho do arquivo adotado. Em cada etapa, após a obtenção das configurações de melhor desempenho, os resultados do WFS serão confrontados com o apresentado diretamente pelo Ext3.

4.4.1 Arquivos de 10MB

No que se refere ao desempenho do WFS operando com arquivos de 10MB, a Figura 4.2 exibe um gráfico com diferença de desempenho entre os 16 cenários de teste do experimento fatorial completo. A configuração-padrão (0000) foi utilizada como base para identificar a diferença de desempenho com relação às demais configurações. É possível notar que as todas as configurações do tipo 1xxx (com o fator A habilitado) apresentaram melhorias desempenho superiores a 2% em relação à configuração-padrão. Já as configurações do tipo 0xxx apresentaram desempenho abaixo das mencionadas anteriormente. Dentre as 1xxx, merecem destaque as configurações 1001, 1011 e 1101, cujos desempenhos foram significativamente superiores, cerca de 4% de ganho em relação a configuração-padrão, enquanto a 1000 apresentou um ganho próximo a 2%. Vale ressaltar também que, para esse tamanho de arquivo, o ganho de desempenho máximo, em

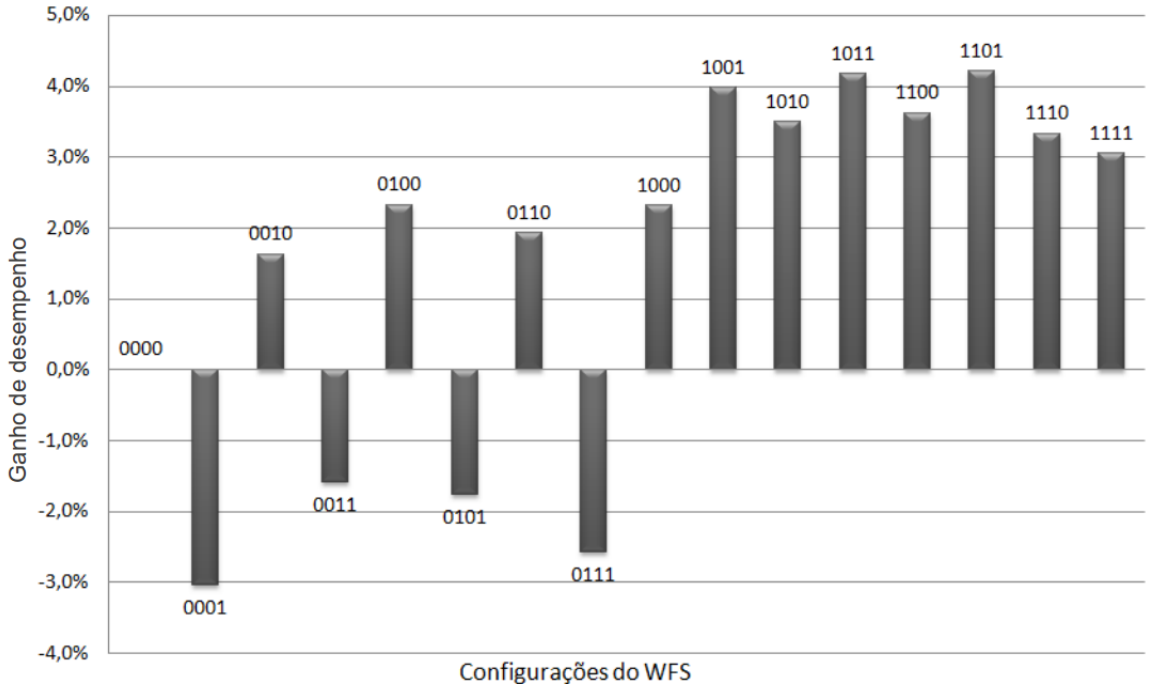

Figura 4.2: Ganho de desempenho em relação à configuração 0000, considerando arquivos de 10MB

relação à 0000, foi de apenas 4%, e que configurações do tipo 0xx1 tiveram desempenhos inferiores aos obtidos pela configuração padrão do sistema, chegando a uma perda de desempenho de até 3%. As configurações 0xx0 apresentaram comportamento médio similar, fornecendo um ganho médio de desempenho na ordem de 2%.

Estes resultados indicam, mas não comprovam, que o *Direct I/O* de montagem (fator A), melhora em até 4% o desempenho do WFS, quando o mesmo está habilitado. Este desempenho é incrementado ainda mais ao habilitar o fator D, mantendo o A acionado. Por outro lado, as comparações iniciais sugerem que as consequências de habilitar o fator D sem ativar o fator A são negativas para a desempenho do WFS, considerando o *benchmark* adotado. Como a configuração-padrão também apresenta pior desempenho que as configurações 1xxx, ela não é indicada.

A Tabela 4.3 fornece as médias aritméticas e desvios padrão para todos os cenários de teste que utilizaram arquivos de 10MB. Assim como pôde ser observado na Figura 4.2, as configurações 1xxx apresentaram os menores tempos de execução, sugerindo que o fator A, de fato, tenha influência na melhoria do desempenho do WFS. Dentre os melhores cenários em relação ao desempenho, ainda merece maior destaque a configuração AD (1001), pois além de boa desempenho, apresenta o mais baixo desvio padrão.

Após a obtenção dos valores médios para cada cenário de teste, cálculos para obter o efeito e a relevância de cada combinação de fatores foram executados, seguindo rigorosamente as técnicas dos experimentos fatoriais. A Tabela 4.4 detalha cada um dos valores

Tabela 4.3: Resultado das medições para o tempo de execução utilizando a adaptação do *Workload Bonnie* para arquivos de 10MB

ABCD	Média (s)	DP (s)
0000	6,40	0,24
0001	6,59	0,32
0010	6,30	0,21
0011	6,50	0,26
0100	6,25	0,22
0101	6,51	0,29
0110	6,28	0,25
0111	6,56	0,33
1000	6,25	0,25
1001	6,14	0,16
1010	6,18	0,22
1011	6,13	0,21
1100	6,17	0,23
1101	6,13	0,23
1110	6,19	0,19
1111	6,20	0,21

obtidos através do experimento fatorial. Cada linha desta tabela refere-se ao efeito e à respectiva relevância de uma junção dos fatores. Vale reforçar que configurações como 1100 agregam os efeitos de todas as combinações entre os fatores A e B, ou seja, o efeito do fator A quando isolado, do B quando isolado e da junção AB. Estes efeitos e a relevância de cada um deles estão dispostos respectivamente nas linhas 1, 2 e 5 da referida tabela.

Como pode ser observado na primeira linha da Tabela 4.4, o fator A apresentou-se como o mais relevante no que se refere à melhoria de desempenho, gerando um efeito de melhoria de desempenho cuja relevância é maior do que 60%. Esse resultado se mostra coerente com o que foi observado, tanto nos valores médios, quanto no gráfico de ganho de desempenho para os arquivos de 10MB, visto que todas as 1xxx apresentam desempenho superiores às 0xxx. Outra junção a ser considerada é a união dos fatores A e D (linha 7), que melhora o desempenho do WFS, apesar de atenuada pelo efeito gerado por D isolado (linha 4). Devido à baixa relevância dos demais fatores, os mesmos foram desconsiderados neste estudo. Assim, como resultado, o experimento fatorial indicou a configuração AD (1001) como a de mais alta desempenho para arquivos de 10MB, dentre as configurações testadas. Uma vez que os demais fatores são irrelevantes, configurações como 1011 e 1101

Tabela 4.4: Efeitos e relevâncias (soma dos quadrados) para os experimentos fatoriais completos utilizando a adaptação do *Workload Bonnie* para arquivos de 10MB

Junção dos fatores	**Efeito (s)**	**Relev. (%)**
A	-0,12	63,66
B	-0,01	0,65
C	-0,01	0,21
D	0,05	9,69
AB	0,01	0,46
AC	0,01	0,25
AD	-0,07	19,90
BC	0,03	3,30
BD	0,02	1,23
CD	0,01	0,40
ABC	-0,01	0,14
BCD	0,00	0,00
ABD	0,00	0,01
ACD	0,01	0,11
ABCD	0,00	0,01

são consideradas apenas variações da 1001.

Após a execução dos experimentos fatoriais, os resultados da configuração 1001 do WFS foram confrontados com o desempenho do sistema de arquivos Ext3 sem características WORM. Para assegurar que tal comparação seja estatisticamente válida, foram realizados testes de normalidade e testes t nas amostras obtidas. Além dos testes, foram calculados o intervalo de confiança e os quartis das amostras.

A Figura 4.3 apresenta o histograma e o resultado do teste de normalidade (Anderson-Darling) da configuração na qual os fatores A e D encontram-se ativados simultaneamente (1001). A partir destas informações, é possível assegurar que o WFS apresenta comportamento semelhante à curva normal, pois o *P-Value* é maior que 0,005, considerando um nível de confiança de 95%. Como a amostra possui um comportamento de acordo com a gaussiana, o teste t pode ser utilizado para comparar os dois resultados estatisticamente. Assim, esta amostra, de fato, representa estatisticamente o desempenho do WFS para arquivos de 10MB.

Uma análise similar à realizada com o WFS foi efetuada sobre o Ext3 diretamente (ver Figura 4.4). No entanto, diferentemente do ocorrido com o sistema WORM, o Ext3 apresentou um comportamento que divergiu da curva normal, uma vez que o *P-*

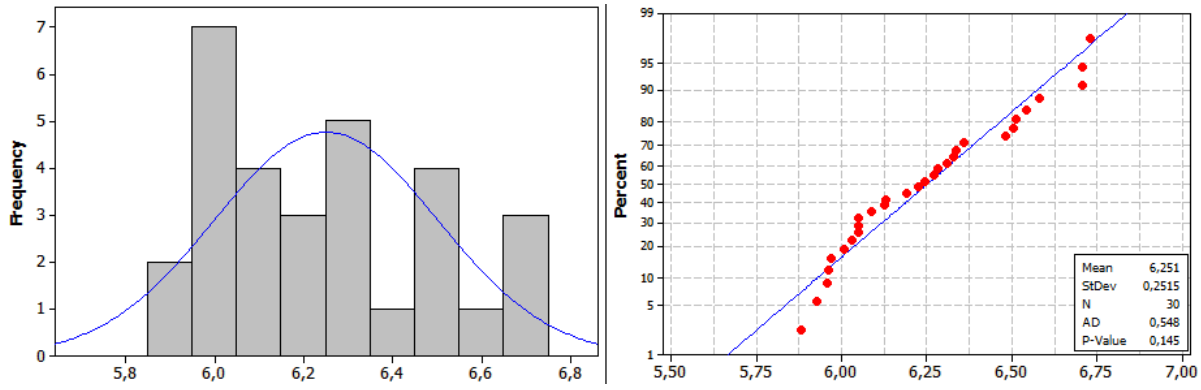

Figura 4.3: Histograma e teste de normalidade para o WFS considerando arquivos de 10MB

Value resultante do seu teste de normalidade foi menor que 0,005. Assim, não foi possível realizar o teste t diretamente na amostra obtida, pois esta não representa estatisticamente o comportamento do Ext3 para arquivos de 10MB.

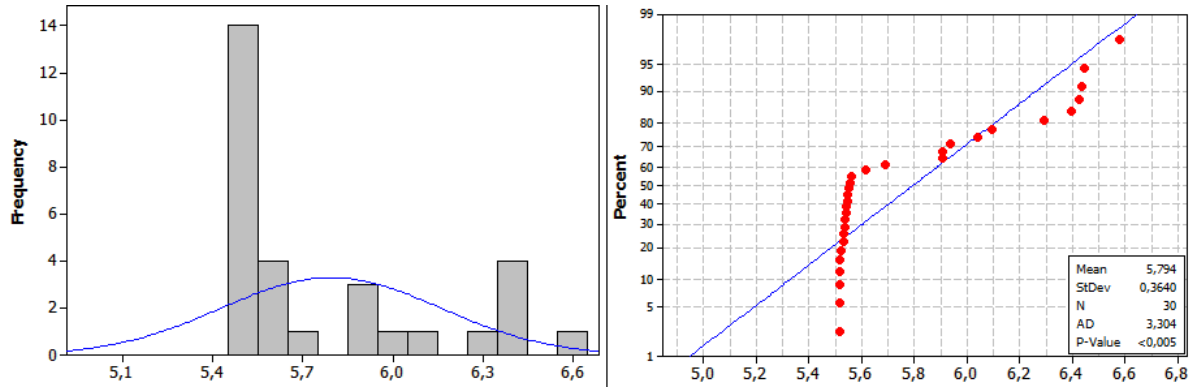

Figura 4.4: Histograma e teste de normalidade para o Ext3 considerando arquivos de 10MB

Para resolver tal situação, o algoritmo de *bootstrap* [ET97] foi aplicado à amostra. Foram efetuadas um total de 30 reamostragens, quantidade suficiente para que o teste de normalidade fosse bem sucedido. A Figura 4.5 traz o histograma e o resultado do teste de normalidade para as amostras do Ext3 não-WORM, após a execução do algoritmo de *bootstrap*.

A Figura 4.6 apresenta o diagrama de caixa das amostras do Ext3 (após o processo de *bootstrap*) e do WFS para arquivos de 10MB. É possível constatar que não há discrepâncias nas amostras e que o tempo de execução do sistema WORM é superior ao alcançado pelo Ext3.

A partir dos resultados estarem de acordo com a curva normal, o teste t pôde ser realizado. Comparações de três tipos foram realizadas com o objetivo de descobrir se existia diferença de desempenho entre os sistemas de arquivos, e se algum deles possuía

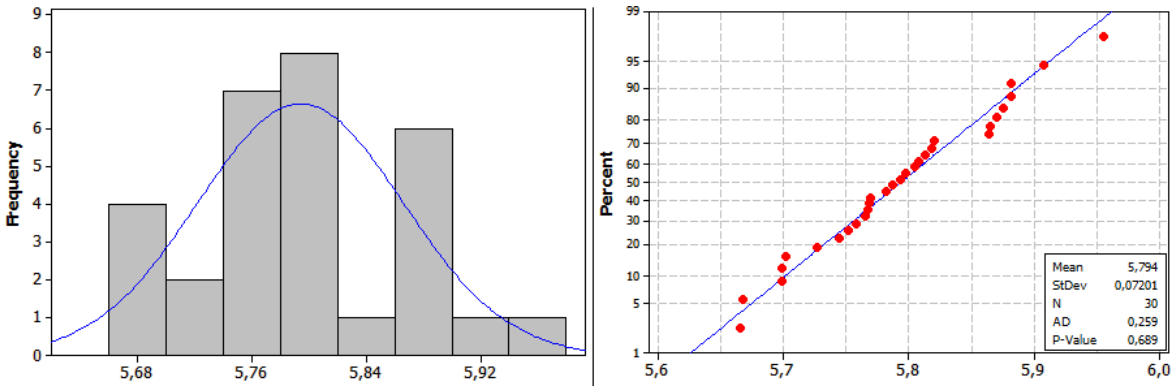

Figura 4.5: Histograma e teste de normalidade para o Ext3 considerando arquivos de 10MB após a execução do bootstrap

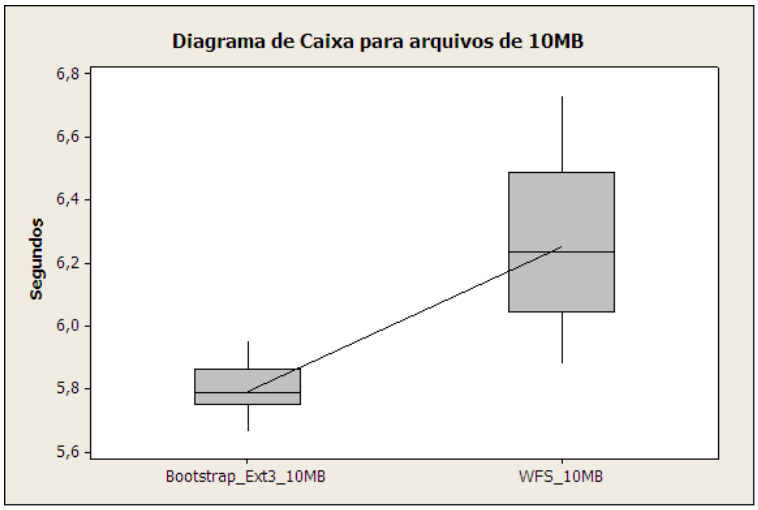

Figura 4.6: Diagrama de caixa para as amostras do Ext3 e do WFS para testes com arquivos de 10MB

desempenho que pudesse ser considerada superior ou inferior ao outro. O resultado desta comparação estatística, obtido a partir do Minitab 15, encontra-se disposto na Figura 4.7. O teste t confirmou a existência de diferença estatística significativa entre as duas amostras, além de assegurar que o WFS apresenta uma pequena perda de desempenho em relação ao Ext3 não-WORM.

A Tabela 4.5 apresenta uma comparação entre o Ext3 e a configuração 1001 do WFS. Optou-se por incluir também os resultados do Ext3 após o bootstrap, para assegurar que os mesmos não apresentaram divergências nos valores médios para este sistema de arquivos. Vale ressaltar que tanto as distâncias entre os valores máximo e mínimo para o intervalo de confiança quanto a distância interquartil (DiQ) sempre foram significativamente inferiores aos valores médio das amostras, confirmando que as mesmas possuíam pouca variação e que o ambiente de medição estava livre de interferências relevantes. Em-

One-Sample T: Bootstrap_Ext3_10MB; WFS_10MB

Variable	N	Mean	StDev	SE Mean	95% Upper Bound
Bootstrap_Ext3_10MB	30	5,7940	0,0720	0,0131	5,8164
WFS_10MB	30	6,2510	0,2515	0,0459	6,3290

One-Sample T: Bootstrap_Ext3_10MB; WFS_10MB

Variable	N	Mean	StDev	SE Mean	95% CI
Bootstrap_Ext3_10MB	30	5,7940	0,0720	0,0131	(5,7671; 5,8209)
WFS_10MB	30	6,2510	0,2515	0,0459	(6,1571; 6,3449)

One-Sample T: Bootstrap_Ext3_10MB; WFS_10MB

Variable	N	Mean	StDev	SE Mean	95% Lower Bound
Bootstrap_Ext3_10MB	30	5,7940	0,0720	0,0131	5,7717
WFS_10MB	30	6,2510	0,2515	0,0459	6,1730

Figura 4.7: Resultados dos testes t para arquivos de 10MB

bora o sistema WORM apresente desempenho inferior ao Ext3 nativo, é possível notar, conforme exposto na Tabela 4.5, que, para os arquivos menores, o Ext3 foi apenas 5,5% mais rápido que o WFS. Esse resultado indica que a junção WFS/FUSE gera apenas uma leve perda de desempenho para o usuário do sistema proposto neste trabalho.

Tabela 4.5: Comparação de tempo de execução entre o WFS e o Ext3 não-WORM, considerando arquivos de 10MB

Sistema de arquivos	Média (s)	DP (s)	95% IC (s)	DiQ (s)
WFS_1001	6,14	0,16	(6,16 ; 6,35)	0,45
Ext3 não-WORM	5,79	0,36	(5,66 ; 5,93)	0,52
Ext3 não-WORM (com bootstrap)	5,79	0,07	(5,77 ; 5,82)	0,11

4.4.2 Arquivos de 1000MB

Em relação aos arquivos de 1000MB, as diferenças de desempenho entre os cenários de teste do experimento fatorial foram mais significativas, conforme exposto na Figura 4.8. Ao executar o *Bonnie* considerando arquivos maiores, o comportamento do WFS apresentou algumas diferenças se comparado aos resultados obtidos anteriormente.

Embora o fator A habilitado eleve consideravelmente o ganho de desempenho, vale destacar que, diferentemente do observado em arquivos de 10MB, praticamente não existe diferença entre o desempenho das 8 configurações do tipo 1xxx. Outro diferencial importante é a melhoria de desempenho, visto que as configurações 1xxx conseguem 23% de ganho em relação à 0000 trabalhando com arquivos maiores. Destaca-se também a melhoria de desempenho das configurações 0xx1, uma vez que, para os arquivos menores,

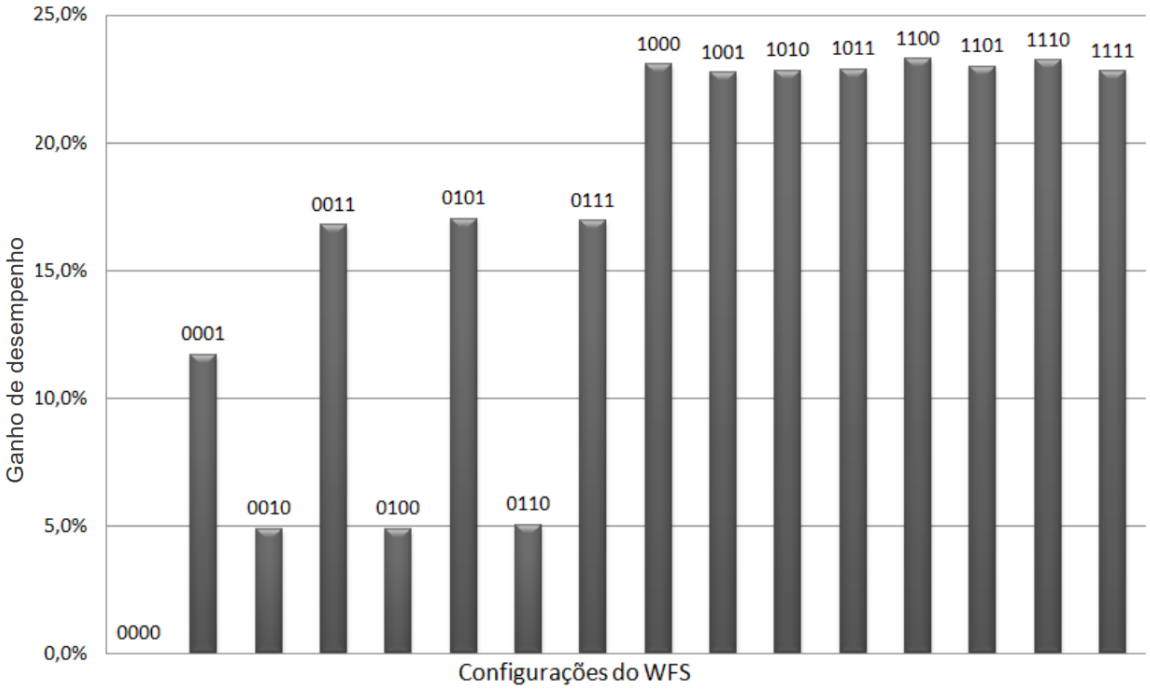

Figura 4.8: Ganho de desempenho em relação à configuração 0000, considerando arquivos de 1000MB

apresentaram tempo de execução inferior e, para os arquivos de 1000MB, demonstram um desempenho cerca de 17% melhor que a configuração 0000, enquanto as outras configurações do tipo 0xx0 apresentam um discreta melhoria de desempenho, na ordem de 5%.

A Tabela 4.6 apresenta a média e o desvio padrão para cada um dos cenários de teste, considerando arquivos de 1000MB. Esta tabela reforça a diferença significativa de desempenho entre as configurações 1xxx e a 0000, indicada pela Figura 4.8. Para os arquivos deste 1000MB, a configuração-padrão do WFS apresentou um tempo de execução significativamente superior às demais. As configurações com o fator A habilitado (1xxx) apresentam os melhores desempenhos, sendo eles bastante próximos entre si. Estes resultados possibilitam que o usuário opte por qualquer das configurações 1xxx com o mesmo desempenho, pois não há evidência de diferença de desempenho entre tais cenários. No entanto, para escolher a melhor configuração dos fatores para o WFS, o efeito e a relevância de cada um deles devem ser calculados.

A Tabela 4.7 apresenta o efeito e a relevância de cada combinação dos fatores, considerando arquivos de 1000MB. Assim como nos arquivos menores, o fator A isolado produz um efeito de quase 70% de relevância, tornando-se, mais uma vez, fundamental para a melhoria de desempenho do WFS. Contudo, para arquivos maiores, os efeitos D e AD, apesar de relevantes (13,2 e 14,32, respectivamente), não devem ser considerados, visto que o efeito produzido pela junção dos fatores A e D anula o efeito benéfico que o fator D propicia isoladamente. Como os demais efeitos possuem baixa relevância, mais uma

Tabela 4.6: Resultado das medições para o tempo de execução utilizando a adaptação do *Workload Bonnie* para arquivos de 1000MB

ABCD	Média (s)	DP (s)
0000	181,31	4,35
0001	160,11	3,66
0010	172,49	3,41
0011	150,91	3,02
0100	172,48	3,74
0101	150,44	2,36
0110	172,17	3,23
0111	150,60	2,48
1000	139,50	2,28
1001	140,07	2,37
1010	139,94	3,01
1011	139,82	2,31
1100	139,11	2,33
1101	139,66	2,77
1110	139,21	2,16
1111	140,00	2,31

vez, foram desconsiderados desse estudo. Assim, o experimento fatorial corrobora com o que foi destacado a partir da Tabela 4.6 e confirma a configuração A (1000) como sendo a de melhor desempenho, considerando as demais 1xxx como variações da original.

Os testes de normalidade foram efetuados tanto sobre o Ext3 quanto sobre o WFS e, diferentemente da análise anterior, ambos indicaram resultados que seguem a distribuição gaussiana (ver Figuras 4.9 e 4.10). Portanto, tanto as amostras do WFS e quanto do Ext3 representam estatisticamente seus respectivos sistemas de arquivos. Assim, o teste t pôde ser aplicado para comparar o desempenho dos sistemas.

A Figura 4.11 apresenta o diagrama de caixa comparativo das amostras do Ext3 e do WFS para arquivos de 1000MB. Da mesma forma que para arquivos de 10MB, constatou-se que não há discrepâncias nas amostras e que o tempo de execução do sistema WORM é superior ao alcançado pelo Ext3.

A Figura 4.12 traz um trecho da tela da ferramenta Minitab 15, no qual os resultados dos testes t foram descritos. Os testes foram realizados para verificar se a primeira amostra é menor ou maior e se realmente existe diferença estatística em relação à segunda. Estes testes confirmaram que há diferença estatística entre o desempenho dos sistemas de

Tabela 4.7: Efeitos e relevâncias (soma dos quadrados) para os experimentos fatoriais completos utilizando a adaptação do *Workload Bonnie* para arquivos de 1000MB

Junção dos fatores	Efeito (s)	Relev. (%)
A	-12,07	68,74
B	-1,28	0,77
C	-1,10	0,57
D	-5,29	13,18
AB	1,11	0,58
AC	1,17	0,65
AD	5,51	14,32
BC	1,13	0,61
BD	0,00	0,00
CD	-0,02	0,00
ABC	-1,10	0,57
BCD	0,11	0,01
ABD	0,11	0,01
ACD	-0,03	0,00
ABCD	0,00	0,00

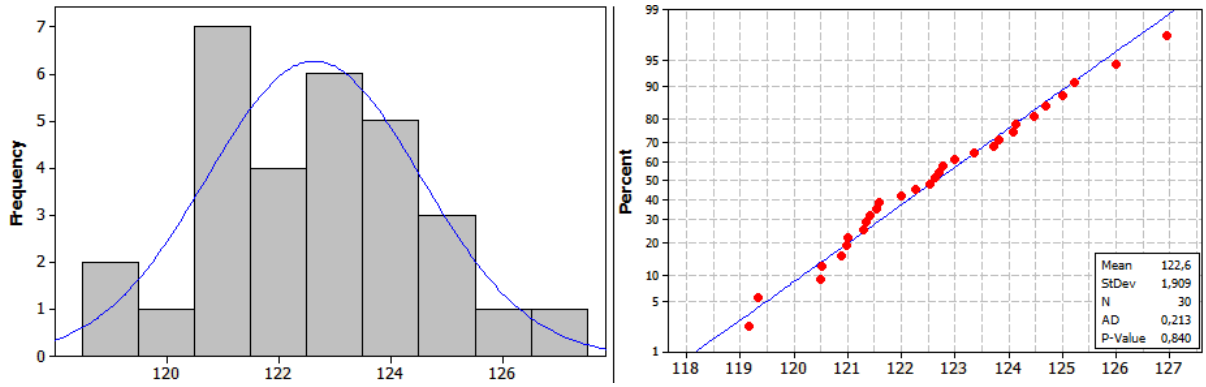

Figura 4.9: Histograma e teste de normalidade para o Ext3 considerando arquivos de 1000MB

arquivos, confirmando, assim, que o Ext3 apresenta desempenho superior ao alcançado pelo WFS.

A Tabela 4.8 apresenta uma comparação dos resultados entre a configuração 1001 do WFS e o Ext3. Antes de discorrer sobre a diferença de desempenho dos dois sistemas, é importante ressaltar que as duas amostras possuem pouca variação nos extremos do

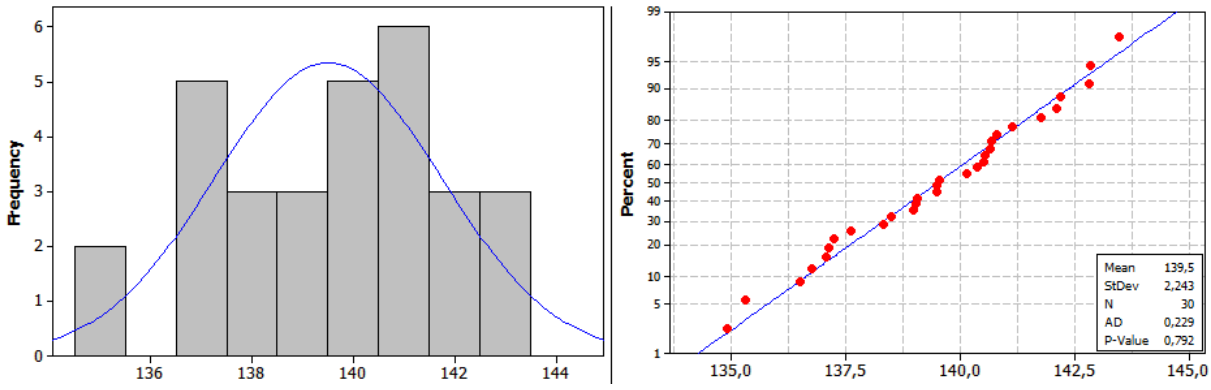

Figura 4.10: Histograma e teste de normalidade para o WFS considerando arquivos de 1000MB

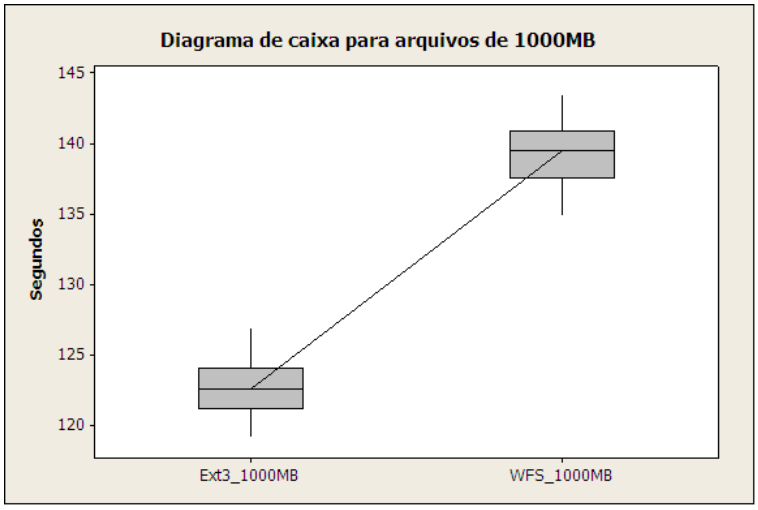

Figura 4.11: Diagrama de caixa para as amostras do Ext3 e do WFS para testes com arquivos de 1000MB

intervalo de confiança e na distância entre interquartis, assim, estão livres de interferência significativa. Com relação à diferença de desempenho entre os sistemas considerando arquivos de 1000MB, o Ext3 foi 12% mais rápido que o WFS. Esse resultado sugere, então, que a utilização do WFS com os dois fatores de *Direct I/O* habilitados trará uma perda de desempenho que atende aos requisitos do sistema, especificados na Seção 3.1.1.

4.5 RESULTADOS PARA AS MÉTRICAS ORIGINAIS DO BONNIE

Como a avaliação de desempenho apresentada neste trabalho tem como foco principal mensurar a diferença de desempenho entre o WFS e o Ext3 nativo do Linux e, não detalhar os motivos para tal diferença, o tempo de execução pode ser utilizado como métrica principal. No entanto, outras métricas obtidas diretamente pela adaptação do

One-Sample T: Ext3_1000MB; WFS_1000MB

Variable	N	Mean	StDev	SE Mean	95% Upper Bound
Ext3_1000MB	30	122,631	1,909	0,348	123,224
WFS_1000MB	30	139,501	2,243	0,410	140,197

One-Sample T: Ext3_1000MB; WFS_1000MB

Variable	N	Mean	StDev	SE Mean	95% CI
Ext3_1000MB	30	122,631	1,909	0,348	(121,919; 123,344)
WFS_1000MB	30	139,501	2,243	0,410	(138,663; 140,338)

One-Sample T: Ext3_1000MB; WFS_1000MB

Variable	N	Mean	StDev	SE Mean	95% Lower Bound
Ext3_1000MB	30	122,631	1,909	0,348	122,039
WFS_1000MB	30	139,501	2,243	0,410	138,805

Figura 4.12: Resultados dos testes t para arquivos de 1000MB

Tabela 4.8: Comparação para o tempo de execução entre o WFS e o Ext3 não-WORM, considerando arquivos de 1000MB

Sistema de arquivos	Média (s)	DP (s)	95% IC (s)	DiQ (s)
WFS_1000	139,50	2,24	(138,7 ; 140,3)	3,38
Ext3 não-WORM	122,63	1,91	(121,9 ; 123,3)	2,87

Bonnie serão expostas nesta seção, com o objetivo único de associar os valores medidos aos respectivos sistemas de arquivos.

Para facilitar o entendimento das tabelas, além de reduzir o tamanho das mesmas, algumas métricas originais do *Bonnie* foram omitidas. Vale ressaltar que, neste estudo, as métricas adotadas visam apresentar as diferenças de comportamento entre os cenários de teste. Assim, apenas as métricas que apresentam tais diferenças de comportamento estão dispostas nas tabelas a seguir. São elas: as taxas de escrita e de leitura de blocos (funções *write* e *read*, respectivamente) e o resultado das buscas concorrentes utilizando funções de *lseek*.

Levando em consideração os arquivos de 10MB, a Tabela 4.9 apresenta os valores das métricas originais do *Bonnie* tanto para o Ext3 quanto para todos os cenários de teste do WFS. Em relação às operações de Escrita, o Ext3 apresentou uma taxa cerca de três vezes superior às maiores medidas no sistema WORM. Também vale ser destacada a diferença dos valores de escrita entre as configurações 1xxx e 0xxx do WFS. As primeiras apresentam taxas cuja magnitude média supera o dobro do obtido pelas configurações cujo fator A apresentava-se desabilitado durante suas medições.

As taxas de leitura apresentaram variações diferenciadas em relação às de escrita. As configurações 0010, 0100 e 0110 apresentaram taxas de escritas elevadas, superiores,

Tabela 4.9: Métricas obtidas pela adaptação do *Workload Bonnie* para arquivos de 10MB

FS	Escrita (K/s)	Leitura (K/s)	Busca (s^{-1})
Ext3	334.811,3	1.846.082,5	127.260,7
WFS 0000	44.821,6	389.623,7	76.970,4
WFS 0001	41.901,1	259.451,6	41.071,6
WFS 0010	47.089,1	2.040.189,4	123.620,0
WFS 0011	50.187,1	307.236,8	42.219,8
WFS 0100	44.423,7	2.023.866,8	128.598,2
WFS 0101	47.665,9	263.231,0	42.770,3
WFS 0110	49.033,0	2.012.254,7	126.778,0
WFS 0111	46.078,5	270.432,5	42.475,9
WFS 1000	116.095,7	287.181,6	38.441,6
WFS 1001	111.401,6	305.754,3	38.216,8
WFS 1010	120.586,3	296.125,3	38.269,3
WFS 1011	116.638,4	319.405,3	39.280,4
WFS 1100	109.081,9	285.496,4	38.160,3
WFS 1101	111.887,8	261.452,5	38.746,0
WFS 1110	112.370,6	301.569,5	38.362,6
WFS 1111	116.562,5	283.703,8	39.844,3

inclusive, à obtida pelo Ext3. As demais configurações (0000, 0xx1 e 1xxx) apresentaram taxas consideravelmente mais baixas, magnitude até 7 vezes inferior. Já as operações de buscas apresentaram um comportamento muito semelhante ao cenário de leitura, visto que as configurações 0010, 0100 e 0110 possuem os valores médios mais altos, similares ao alcançado pelo Ext3 (127.260,7). Já as demais configurações (0001, 0011, 0101, 0111 e 1xxx) alcançaram índices consideravelmente mais baixos (em torno de 39.000). Contudo, apenas a configuração-padrão conseguiu superar essa taxa, atingindo quase 77.000 buscas por segundo.

Ao confrontar os resultados obtidos para as métricas originais do *Bonnie* com o desempenho apresentado por cada sistema de arquivos é possível apontar indícios que relacionam a diferença de desempenho entre os sistemas. Contudo, para confirmarmos tais afirmações, um estudo mais profundo seria necessário. No entanto, como não é o propósito desta avaliação de desempenho oferecer essas garantias, apenas iremos citar algumas hipóteses:

- O Ext3 apresenta melhor desempenho que as configurações do WFS, pois apresenta o maior valor médio para as taxas de escrita e de leitura, assim como operações de

busca;

- As configurações 0xx1 apresentam pior desempenho que a configuração-padrão, pois apresentam taxas de escrita, de leitura e de busca inferiores às apresentadas pela configuração 0000;

- As configurações 1xxx do WFS apresentam-se como as melhores alternativas em relação ao desempenho, pois possuem taxas de escrita superiores, na ordem de duas vezes, às apresentadas pelas configurações 0xxx.

A Tabela 4.10 mostra os resultados das métricas obtidas através do *Bonnie* operando com arquivos de tamanho igual a 1000MB. A taxa de escrita apresentou uma variação de magnitude dos valores médios inferior a 10%. Essa variação foi significativamente menor que o apresentado pelos arquivos de 10MB. Para tal métrica, o Ext3 não apresenta superioridade em relação ao WFS. Pelo contrário, algumas configurações conseguiram taxas de escrita levemente superiores ao sistema nativo do Linux.

Contudo, ao serem comparadas as taxas de leitura, o Ext3 apresenta vantagem, pois possui um valor médio com magnitude cerca de duas vezes maior que os melhores cenários do WFS, atingindo quase 230.000K/s. É importante destacar que as configurações 1xxx possuem os melhores valores dentre as configurações do WFS, superando os 110.000 K/s de leitura, enquanto a configuração-padrão teve média levemente superior aos 40.000K/s. É possível notar que as configurações do tipo 0xx1 apresentam taxas de leitura de magnitude cerca de duas vezes superiores às 0xx0. No entanto, vale ser reforçado que as configurações 0xx1 apresentam desempenho de leitura inferior às configurações com o fator A habilitado.

Com relação à realização de buscas concorrentes, praticamente não existe diferença entre os valores alcançados com o Ext3 e o WFS 1xxx, apresentando o sistema WORM, em algumas situações, valores mais elevados para essa métrica. As configurações do tipo 0xxx, alcançaram valores de magnitude na ordem de 3 vezes menor que o Ext3 e o WFS 1xxx.

De forma similar ao resultado anterior, confrontando os valores das métricas obtidos diretamente pelo *Bonnie* com o desempenho alcançado pelos sistemas de arquivos, algumas hipóteses para tal diferença de desempenho puderam ser elaboradas. São elas:

- O Ext3 apresenta desempenho melhor que as configurações do WFS, pois apresenta o maior valor médio para a taxa de leitura, bem como para as buscas concorrentes;

- As configurações 0xx1 modificaram o comportamento em relação aos arquivos de 10MB, pois, para arquivos de 1000MB, apresenta melhor desempenho que as configurações 0xx0. Isso pode ser consequência da melhor da taxa de leitura apresentada pelas configurações 0xx1 em relação às 0xx0. Vale ser ressaltado que, neste estudo, não foram constatadas diferenças significativas entre as taxas de escrita e os resultados das operações de busca concorrentes, considerando as configurações 0xxx;

Tabela 4.10: Métricas obtidas pela adaptação do *Workload Bonnie* para arquivos de 1000MB

FS	Escrita (K/s)	Leitura (K/s)	Busca (s^{-1})
Ext3	31.822,0	227.429,0	1.773,8
WFS 0000	34.043,9	41.347,3	501,8
WFS 0001	33.693,3	102.891,9	503,5
WFS 0010	31.458,9	39.791,4	501,6
WFS 0011	31.988,7	83.144,6	497,1
WFS 0100	31.687,6	39.694,5	501,3
WFS 0101	32.140,4	82.043,1	496,4
WFS 0110	32.249,4	40.145,1	500,3
WFS 0111	32.456,1	82.880,0	499,2
WFS 1000	31.921,7	117.016,0	1.798,2
WFS 1001	31.107,5	118.970,4	1.766,1
WFS 1010	31.697,8	117.648,6	1.780,8
WFS 1011	31.373,7	117.321,6	1.800,9
WFS 1100	31.923,7	117.497,2	1.804,2
WFS 1101	31.993,8	114.811,7	1.780,2
WFS 1110	32.153,1	116.750,8	1.795,0
WFS 1111	31.706,6	115.809,1	1.805,1

- As configurações 1xxx do WFS apresentam-se como as melhores alternativas em relação ao desempenho, pois possuem a taxa de leitura e o desempenho de busca concorrente significativamente superiores aos valores obtidos por configurações 0xxx. Deve ser destacado o resultado obtido nas operações de busca, que chegaram aos 1.800,0, superando, em muitos casos, o valor apresentado pelo Ext3, enquanto as configurações 0xxx apresentaram média próxima a 500,0.

4.6 CONSIDERAÇÕES FINAIS

Para ambos os tamanhos de arquivo utilizado pelo *Bonnie* adaptado, a opção de *Direct I/O* habilitada no processo de montagem do sistema de arquivos (fator A) propiciou os melhores resultados de desempenho para o WFS. A partir dos resultados obtidos através dos experimentos fatoriais completos, as configurações AD (1001) e A (1000) mostraram-se as mais significativas em relação à melhoria do desempenho do WFS para os arquivos de 10MB e 1000MB, respectivamente. Uma comparação de valor estatístico foi realizada entre o *Write-Once Read-Many File System* e o Ext3 não-WORM, ficando evidenciada, para os cenários testados, uma diferença de desempenho inferior a 20%. Esses resultados

atingem o requisito de desempenho detalhado na Seção 3.1.1. Vale salientar que, ao operar com arquivos de 10MB, a configuração 1000 do WFS apresentou um *overhead* em relação à 1001. Então, o estudo sugere a configuração AD (1001) como a mais indicada para reduzir a perda de desempenho em relação ao Ext3, pois a mesma apresentou bom desempenho em todos os cenários de teste.

CAPÍTULO 5

CONCLUSÕES

Independentemente do conteúdo armazenado nos computadores, a perda de dados, seja ela por remoções involuntárias ou por problemas no *hardware*, geram uma série de inconvenientes para os usuários. Sistemas operacionais convencionais oferecem mecanismos de proteção insuficientes para evitar este transtorno. A imutabilidade pode ser utilizada para minimizar tal problema. Entende-se como imutabilidade a propriedade que, uma vez aplicada a um objeto, proíbe qualquer modificação subsequente a este. Esta é a propriedade que norteia os dispositivos *Write-Once Read-Many*, que têm sido historicamente utilizados para arquivamento de dados que requerem um longo período de conservação.

A política *Write-Once Read-Many*, quando aplicada a um dispositivo de armazenamento, permite que dados sejam criados, mas impedem que os mesmos sejam modificados ou removidos. Tendo como base esta política, o presente trabalho propõe um sistema de arquivos, no espaço do usuário, cujo principal objetivo é reduzir o risco de perda de informações causadas por remoção ou alteração de conteúdo involuntárias, sem que, para isso, seja necessário um investimento em *hardware* de alto custo. Denominada de WFS, esta ferramenta também permite aos usuários trabalhar com cópias dos dados originais, movendo-as para outros diretórios, realizando *backups*

O WFS foi implementado utilizando a infraestrutura FUSE, que possibilita a portabilidade entre diversos sistemas operacionais, além de outras vantagens, como a facilidade de desenvolvimento e a possibilidade de utilização (e montagem) por parte de usuários sem privilégios de administrador. As principais características presentes neste sistema são: ser código-livre; atender à usuários sem privilégios de administração; possuir rápido processo de instalação e de configuração; apresentar desempenho satisfatório; possibilitar uma forma alternativa de modificação e de remoção de dados e permitir o gerenciamento dos dados de forma simples.

Adicionalmente, um mecanismo de *trace* foi desenvolvido para o WFS. Quando habilitado, ele cria um arquivo de log com o histórico das funções do WFS, na ordem em que foram postas em execução pelo FUSE. Argumentos provenientes do FUSE podem ser adicionados facilmente ao *trace*, assim como mensagens para depuração de erros. A partir da utilização desse mecanismo, gargalos de desempenho de uma versão preliminar do WFS foram descobertos.

A avaliação de desempenho dessa ferramenta foi executada com o objetivo de mensurar a perda de desempenho em relação a um sistema de arquivos nativo Linux sem características WORM, o Ext3. Para tal, um experimento fatorial completo foi realizado, indicando a existência de diferença estatística significativa no desempenho do WFS, de acordo com o ajuste dos fatores utilizados no estudo. Este resultado sugere que, ao uti-

lizar o WFS com os fatores de *Direct I/O* de montagem e de código-fonte habilitados, essa perda é minimizada, fazendo com que o sistema WORM apresente uma diferença de desempenho em relação ao Ext3 que não deve ser suficiente para desestimular a utilização do sistema proposto por usuários residenciais.

5.1 CONTRIBUIÇÕES

As principais contribuições deste trabalho são:

- um sistema de arquivos de código-livre que pode eliminar a perda de dados por remoções/alterações de conteúdo involuntárias, o WFS. Este sistema permite que o usuário possa usufruir de suas funcionalidades, mesmo sem privilégios de administrador. Vale ressaltar que o WFS pode ser executado em diversos sistemas operacionais e não necessita que o usuário adquira novos equipamentos para utilizá-lo.

- um levantamento de parâmetros de configuração do FUSE foi realizado durante a etapa de avaliação de desempenho. Quatro desses fatores foram apresentados e discutidos. O estudo confirmou que o ajuste desses parâmetros de configuração influencia no desempenho do WFS;

- o resultado da avaliação de desempenho realizada demonstrou que o WFS-FUSE apresenta apenas uma discreta perda de desempenho em relação ao Ext3, para a carga de trabalho utilizada.

5.2 LIMITAÇÕES

As limitações deste trabalho são as seguintes:

- a utilização do WFS não elimina a necessidade de efetuar cópias de segurança dos dados, pois, apesar de assegurar que os dados já armazenados não serão alterados ou removidos quando acessados diretamente pela proteção WORM, não é possível impedir que danos físicos ao equipamento ou corrupção de dados ocorram;

- como o WFS é voltado para usuários residenciais, em que o risco de ameaças contra a segurança do ambiente é bastante reduzido, optou-se por não inserir mecanismos de proteção contra invasão. Assim, evitou-se que o desempenho do sistema fosse afetado pela inserção de um mecanismo que, na maioria dos casos, não seria utilizado por estes usuários;

- por se tratar de um sistema de arquivos no espaço do usuário, o WFS não é o responsável direto por operar os dados em disco, devendo, então, repassar as solicitações de I/O para o sistema de arquivos nativo do SO. Para a realização desta

atividade, o FUSE necessita efetuar trocas de contexto e cópias de conteúdo adicionais, acarretando leve perda de desempenho. Vale ressaltar que um sistema de arquivos localizado no *kernel* do SO poderia apresentar desempenho, até mesmo, superior aos tradicionais R/W, visto que suas estruturas de dados poderiam ser otimizadas por não permitirem alteração nos conteúdos após serem armazenados;

- dentre os requisitos do sistema, elicitados a partir das reuniões semanais com o cliente, o WFS apresenta uma forma alternativa de alteração e de remoção de conteúdo. Isto pode ser realizado através da pasta original do sistema de arquivos nativo. Contudo, vale ressaltar que, sobre o referido diretório, não há nenhuma proteção contra remoções ou alterações involuntárias providas pelo WFS, apenas as convencionais do sistema operacional.

5.3 TRABALHOS FUTUROS

Como trabalhos futuros, podemos destacar:

- acrescentar à versão atual do WFS mecanismos de integridade e de proteção de acesso aos dados. Estas funcionalidades podem ser potencialmente úteis para arquivos críticos dos usuários, normalmente referentes à documentação financeira ou comercial. Contudo, a utilização desse recurso deve ser fornecida ao usuário de opcional;

- adicionar mecanismos de autenticação e criptografia que gerenciem o acesso aos dados armazenados, fornecendo maior nível de segurança e de privacidade ao usuário do WFS;

- implementar um mecanismo de cópia de segurança que, a partir de uma solicitação simples do usuário, realize o *backup* dos dados gerenciados pelo WFS em CD/R, DVD/R ou discos removíveis;

- desenvolver uma versão do WFS para o contexto e para as necessidades de corporações, localizada no *kernel* do Linux. Garantias de desempenho, de integridade, de segurança e de retenção de dados devem ser fornecidas. Assim, esta versão seria voltada para aplicações mais críticas. Uma comparação de desempenho entre as versões do WFS, tanto no *kernel* quanto no espaço do usuário, poderia ser realizada.

REFERÊNCIAS BIBLIOGRÁFICAS

[ACF01] C.J. Antonelli, K.W. Coffman, and J.B. Fields. The 10 Mbps Advanced Packet Vault. 2001.

[Bel09] Labs Bells. Plan9. http://plan9.bell-labs.com/plan9/, 2009.

[Blo07] Mathieu Blondel. Wikipediafs. http://wikipediafs.sourceforge.net/, 2007.

[Bra10] T. Bray. Bonnie benchmark tool. http://www.textuality.com/bonnie/, 2010.

[CDVD+03] D. Clarke, S. Devadas, M. Van Dijk, B. Gassend, and G. Suh. Incremental multiset hash functions and their application to memory integrity checking. *Advances in Cryptology-ASIACRYPT 2003*, pages 188–207, 2003.

[Chr94] Panagiotis Christias. Time. http://unixhelp.ed.ac.uk/CGI/man-cgi?time, 1994.

[Cor10a] Data Archive Corporation. About udo (ultra density optical). http://www.dataarchivecorp.com/about-udo.htm, 2010.

[Cor10b] EMC Corporation. Emc centera. http://www.emc.com/products/family/emc-centera-family.htm, 2010.

[Cor10c] EMC Corporation. Emc corporation. http://www.emc.com/, 2010.

[Cor10d] IBM Corporation. Ibm totalstorage enterprise. http://www-03.ibm.com/servers/storage/, 2010.

[dRFdB98] Presidência da República Federativa do Brasil. Lei 9.610. http://www.planalto.gov.br/ccivil/leis/L9610.htm, 1998.

[Dut02] S. Dutta. Use direct i/o to improve performance of your aix applications. http://www.ibm.com/developerworks/aix/library/au-DirectIO.html, 2002.

[Eat07] P. Eaton. Fuse, direct_io, mmap, and executables. http://article.gmane.org/gmane.comp.file-systems.fuse.devel/5292, 2007.

[ET97] B. Efron and R.J. Tibshirani. *An introduction to the bootstrap*. Chapman & Hall, 1997.

[Evi04] et al Evi. *Manual Completo do Linux - Guia do Administrador*. Makron Books, 2004.

[Fal10] T. Falcao. Wfs: A write-once read-many file system. sourceforge.net/projects/worm-filesystem, 2010.

[FAM+10a] T. Falcao, E. Andrade, R. Matos, R. Ferraz, P. Maciel, S. Worth, and P. Malenfant. Otimização do Desempenho de um Sistema de Arquivos FUSE-Linux Baseado na Política Write-Once Read-Many . Conferência Latino-americana de Informática, 2010.

[FAM+10b] T. Falcao, E. Andrade, R. Matos, P. Maciel, S. Worth, and P. Malenfant. WFS: Um Sistema de Arquivos FUSE-Linux Baseado na Política Write-Once Read-Many. Workshop de Software Livre, 2010.

[Fed10a] Governo Federal. Computador para todos. http://www.computadorparatodos.gov.br, 2010.

[Fed10b] Governo Federal. Computador portátil para professores. http://www.computadorparaprofessores.gov.br/, 2010.

[FKM00] K. Fu, M.F. Kaashoek, and D. Mazieres. Fast and secure distributed read-only filesystem. *ACM Transactions on Computer Systems*, pages 181–196, 2000.

[FM03] P.J. Fortier and H.E. Michel. *Computer systems performance evaluation and prediction.* Digital Press, 2003.

[Fou10] Free Software Foundation. Gnu general public license version 3. http://www.gnu.org/licenses/gpl-3.0.html, 2010.

[Fur10] L. Furquim. Chironfs: Um sistema de arquivos deve continuar, ainda que danificado. http://www.furquim.org/chironfs/pt/, 2010.

[FUS10a] FUSE. Operating system. http://sourceforge.net/apps/mediawiki/fuse/ index.php?title=OperatingSystems, 2010.

[FUS10b] FUSE. Windows. http://sourceforge.net/apps/mediawiki/fuse/ index.php?title=OperatingSystemsWindows, 2010.

[Gen94] W. Genosa. Monitoring performance with iostat and vmstat. *Sys Admin*, 3(2):6–16, 1994.

[GGL03] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[Gou10] V. Gough. Encfs encrypted filesystem. http://www.arg0.net/encfs, 2010.

[GR+95] S.R. Guido Russo et al. An operating system independent WORM archival system. *Software: Practice and Experience*, 25(5):521–532, 1995.

[Gut94] P. C. Gutmann. Secure filesystem (sfs) for dos/windows. www.cs.auckland.ac.nz/ pgut001/sfs/index.html, 1994.

[Hen10] C. Henk. Fuse for freebsd. http://fuse4bsd.creo.hu/, 2010.

[HSW07] R. Hasan, R. Sion, and M. Winslett. Introducing secure provenance: problems and challenges. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, page 18. ACM, 2007.

[HTS+05] R. Hasan, J. Tucek, P. Stanton, W. Yurcik, L. Brumbaugh, J. Rosendale, and R. Boonstra. The techniques and challenges of immutable storage with applications in multimedia. In *Proc. SPIE*, volume 5682, pages 41–52. Citeseer, 2005.

[HWS07] R. Hasan, M. Winslett, and R. Sion. Requirements of Secure Storage Systems for Healthcare Records. *Lecture Notes in Computer Science*, 4721:174, 2007.

[IBO10] IBOPE. Ibope nielsen online. http://www.ibope.com.br/, 2010.

[Inc10] Linux Online Inc. Linux online. http://www.linux.org/, 2010.

[Inf10] Infowester. Sistema de arquivos ext3. http://www.infowester.com/linext3.php, 2010.

[Jai08] R. Jain. *The Art Of Computer Systems Performance Analysis: Techniques For Experimental Measurement, Simulation, And Modeling*. Wiley India Pvt. Ltd., 2008.

[Kel10] M. Keller. Rofs, the read-only filesystem for fuse. http://mattwork.potsdam.edu/projects/wiki/index.php/Rofs, 2010.

[Kir10] Dustin Kirkland. standards - c and unix standards. http://manpages.ubuntu.com/manpages/intrepid/man7/standards.7.html, 2010.

[LID10] LIDS. Linux intrusion detection system. http://www.lids.org/, 2010.

[Lil05] D.J. Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge Univ Pr, 2005.

[Lin09] LinuxQuestions.org. Extended attributes. http://wiki.linuxquestions.org/wiki/Extended_attributes, 2009.

[Lou08] Phillip Loughe. Squashfs. http://squashfs.sourceforge.net/, 2008.

[Ltd10] Nero Ltd. Nero. http://www.nero.com, 2010.

[Lue03] C. Lueth. WORM Storage on Magnetic Disks Using SnapLock Compliance and SnapLock Enterprise. *Network Applicance, Inc., Sep*, pages 1–5, 2003.

[Mac07] L.P. Machado, F.B.; Maia. *Arquitetura de Sistemas Operacionais*. LTC Editora, 2007.

[Max10] Maxoptix. Magneto optical disks (mo disks). http://www.maxoptix.com/magneto_optical_disks.html, 2010.

[MS01] M. Mitchell and A. Samuel. *Advanced Linux Programming*. New Riders Publishing Thousand Oaks, CA, USA, 2001.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[MW06] S. Mitra and M. Winslett. Secure deletion from inverted indexes on compliance storage. In *Proceedings of the second ACM workshop on Storage security and survivability*, pages 67–72. ACM New York, NY, USA, 2006.

[MWHM07] S. Mitra, M. Winslett, WH Hsu, and X. Ma. Trustworthy Migration and Retrieval of Regulatory Compliant Records. In *24th IEEE Conference on Mass Storage Systems and Technologies, 2007. MSST 2007*, pages 100–113, 2007.

[Net10] NetBSD. The netbsd poject. http://www.netbsd.org/docs/misc/puffs.html, 2010.

[NTF10] NTFS.com. New technology file system. 2010.

[OFK04] J.K. O'Herrin, N. Fost, and K.A. Kudsk. Health Insurance Portability Accountability Act (HIPAA) regulations: effect on medical record research. *Annals of surgery*, 239(6):772, 2004.

[Org10] World Intellectual Property Organization. Copyright and related rights. http://www.wipo.int/copyright/en/, 2010.

[Par09] European Parliament. Legislative documents. http://ec.europa.eu/justice_home/fsj/privacy/ law/index_en.htm, 2009.

[Pav10a] G. Pavarin. Acesso à internet cresce no brasil. http://info.abril.com.br/noticias/internet/acesso-a-internet-cresce-8-2-no-brasil-31032010-19.shl, 2010.

[Pav10b] Igor Pavlov. Lzma. http://www.7-zip.com/sdk.html, 2010.

[PC10] Inc. PowerISO Computing. Poweriso. www.poweriso.com/, 2010.

[QD02] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, 2002.

[Qui02] Daniel Quinlan. Cramfs tool. http://sourceforge.net/projects/cramfs/, 2002.

[Res00] Microsoft Research. Encrypting file system for windows. www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp, 2000.

[RG10] A. Rajgarhia and A. Gehani. Performance and Extension of User Space File Systems. 2010.

[Sil10] A. Silberschatz. *Fundamentos de Sistemas Operacionais*. LTC Editora, 2010.

[Sin10a] A. Singh. Macfuse. http://code.google.com/p/macfuse/, 2010.

[Sin10b] S. Singh. Develop your own filesystem with fuse. http://www.ibm.com/developerworks/linux/library/l-fuse/, 2010.

[Sio08] R. Sion. Strong WORM. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pages 69–76, 2008.

[Sol10] Open Solaris. Fuse on opensolaris. http://www.opensolaris.org/os/project/fuse/, 2010.

[SW07] R. Sion and M. Winslett. Regulatory-compliant data management. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1433–1434. VLDB Endowment, 2007.

[SWZ04] G. Sivathanu, C.P. Wright, and E. Zadok. Enhancing file system integrity through checksums. 2004.

[SWZ05a] G. Sivathanu, C.P. Wright, and E. Zadok. Ensuring data integrity in storage: Techniques and applications. In *Proceedings of the 2005 ACM workshop on Storage security and survivability*, pages 26–36. ACM New York, NY, USA, 2005.

[SWZ05b] G. Sivathanu, C.P. Wright, and E. Zadok. Ensuring data integrity in storage: Techniques and applications. In *Proceedings of the 2005 ACM workshop on Storage security and survivability*, pages 26–36. ACM, 2005.

[Sza10] S. Szakacsits. Ntfs-3g. http://www.tuxera.com/community/ntfs-3g-download/, 2010.

[Sze05] M. Szeredi. Fuse: more flexible caching. http://lkml.org/lkml/2005/9/2/222, 2005.

[Sze09] M. Szeredi. fuse file info struct reference. http://fuse.sourceforge.net/doxygen/, 2009.

[Sze10] M. Szeredi. Filesystem in userspace, 2010.

[Tan09] A.S. Tanenbaum. *Sistemas Operacionais Modernos*. Pearson Education, 2009.

[Twe00] S. Tweedie. EXT3, journaling file system. In *Ottawa Linux Symposium*, pages 24–29, 2000.

[Wil96] R. Williams. P-WORM, E-WORM, S-WORM: Is a sausage a wienie. *Imaging World, July*, 1996.

[WMZ03] C.P. Wright, M. Martino, and E. Zadok. NCryptfs: A secure and convenient cryptographic file system. In *Proceedings of the Annual USENIX Technical Conference*, pages 197–210, 2003.

[WZ03] Y. Wang and Y. Zheng. Fast and secure magnetic WORM storage systems. 2003.

[Yli09] J. Yliluoma. Cromfs: Compressed rom filesystem for linux (user-space). http://bisqwit.iki.fi/source/cromfs.html, 2009.

APÊNDICE A

LICENÇA DE UTILIZAÇÃO DO WFS

O WFS segue a licença *GNU General Public License* versão 3, a GPLv3 [Fou10]. Ela tem como característica principal garantir a liberdade de compartilhar e modificar todas as versões de um programa, desde que ele continue sendo *software* livre para todos os seus usuários. Vale ressaltar que esta licença refere-se à liberdade em relação à distribuição do *software*. Desta forma, o WFS está disponível gratuitamente para utilização e modificação [Fal10].

Assim, baseado na GPLv3, qualquer pessoa está autorizada (e encorajada) a distribuir cópias do WFS, a modificar o *software* ou a usar partes dele em novos programas livres. Como a licença do WFS afirma que não há nenhuma garantia pelo *software* distribuído, o autor e os desenvolvedores estão isentos de qualquer responsabilidade por eventuais danos gerados pela utilização deste *software*. Também deve ser ressaltado que todas as versões modificadas por desenvolvedores devem ser marcadas conforme estabelecido na definição oficial da licença.

APÊNDICE B

CONFIGURAÇÃO DO AMBIENTE

O ambiente de medição utilizado neste estudo encontra-se definido abaixo conforme descrito pelo fabricante (Dell):

- Module, Label, N-SERIES, Small Inspiron
- Base, Notebook, Core Merom T7500, 2.2, 1500
- Processor, T7500, 2.2, 4MB, Core Merom, E1
- Service Install Module Software, VOSTRO, 1500
- MODULE..., KEYBOARD..., 86, BRAZILIAN PORTUGUESE..., 1400/1500, BRAZIL CUSTOMER CENTER...
- Keyboard, 87, Brazil, Single Pointing, Barents, Vostro
- Module, Label, Intel, Notebook Centrino Mobile Technology Santa Rosa Merom
- Ship Group, Notebook Brazil/brazilian, 1500, Brazil Customer Center
- INFORMATION..., NO ITEM
- GUIDE..., OWNER..., VOS, 1500, BRAZILIAN PORTUGUESE...
- Cord, Power, 250V, 2.5A, 1M, C7 Brazil
- PLACEMAT..., GETTING STARTED..., VOS, 1500, DAO/EMEA
- Guide, Product, Information Client, DAO/BCC
- Module, Dual In-line Memory Module, 512, 667MHZ, 1X512, BrazilCustomer Center
- Module, Dual In-line Memory Module, 2048, 667MHZ, 1X2048, BrazilCustomer Center
- Dual In-Line Memory Module 512MB, 667, 64X64, Y9525, Brazil Customer Center
- Module, Battery, Primary, 85WHR 9C, Panasonic
- Battery, Primary, 85WHR, 9C Lithium, Panasonic

- Module, Dvd+/-rw, Hitachi Lg Data Storage, CGSYD, VOS
- Assembly, Dvd+/-Rw, 8X, IDE Hitachi Lg Data Storage Corsica/Gilligan/Sapporo/Yebis
- Module, Software, Roxio, Creator Dell Edition, 9.0, TransactionalLine of Business
- Module, Liquid Crystal Display 15.4WXGA+, True Life, Lg PhilipsLcd, 1500
- Cover, Screw, Mylar, Black, 1500 1700
- Assembly, Cable, Printed CircuitBoard, Liquid Crystal Display Corsica/Sapporo/Lions Gate
- Liquid Crystal Display 15.4WXGA+, VIDEO ELEC. STDS. ASSOC...., True Life Lg Philips Lcd
- Module, Liquid Crystal Display Cover, BLACK, No-camera, 1500
- Bezel, Plastic, Liquid Crystal Display, Black, 1500, No-Camera
- Assembly, Microphone, IntegratedNotebook, Liteon, Mobile 2008
- Module, Adapter, Alternating Current, 90W, Liteon, Power Factor Correction, World Wide
- Assembly, Adapter, Alternating Current, 90W, MOBILE 2007..., Lead Free Liteon
- Module, Card, Graphics, NVIDA 128MB, G86
- Card, Graphics, NVIDA, 128MB G86,
- Module, Assembly, Base, Discrete 1500
- Screw, M3X3, K SCREW HEAD..., MICROSOFT..., BLACK OXIDE...
- Assembly, Base, Notebook Discrete, 1500
- Module, Hard Drive, 160G, 5.4K Small Form Factor, SA080, VOS
- HARD DRIVE..., 160G, Serial ATA..., 9.5, 5.4, SA080
- Module, Software, Free Dos English, Notebook, World Wide
- kit, Software, Free DOS CD/Document
- Module, Media, Digital Video Disk Drive, Resource Dvd, N-SRS 1500
- Module, Card, Network, 3945 Inspiron, Most Of World 2
- Card, Network, Minicard, 3945ULD Most Of World 2
- Bumper, Liquid Crystal Display Rubber, Black, 1500

- Assembly, Cover, Back, 15.4 Liquid Crystal Display, Black 1500
- Heatsink, Central Processor Unit, Notebook, Discrete, Santa Rosa Merom, 1520
- Assembly, Cover, Hinge, Plastic 1500

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).