Instituto Federal Catarinense
Bacharelado em Ciência da Computação
Campus Blumenau

ÁLVARO ALVIN OESTERREICH SANTOS

Sistema de *Tagging* Integrado ao Sistema de Arquivos

Blumenau

2023

ÁLVARO ALVIN OESTERREICH SANTOS

Sistema de *Tagging* Integrado ao Sistema de Arquivos

Trabalho de Conclusão de Curso, submetido ao Curso de Ciência da Computação do Instituto Federal Catarinense – *Campus Blumenau* para a obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Ricardo de la Rocha Ladeira, Me.
Coorientador: Prof. Éder Augusto Penharbel, Me.

Blumenau

2023

ÁLVARO ALVIN OESTERREICH SANTOS

SISTEMA DE *TAGGING* INTEGRADO AO SISTEMA DE ARQUIVOS

Este Trabalho de Conclusão de Curso, foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Bacharelado em Ciência da Computação do Instituto Federal Catarinense – *Campus* Blumenau.

autenticação eletrônica na folha de assinaturas

Prof. Ricardo de la Rocha Ladeira, Me.

Orientador – IFC campus Blumenau

Prof. Éder Augusto Penharbel, Me.

Coorientador – IFC campus Blumenau

BANCA EXAMINADORA

autenticação eletrônica na folha de assinaturas

Prof. Hylson Vescovi Netto, Dr.

IFC campus Blumenau

autenticação eletrônica na folha de assinaturas

Prof. Paulo Cesar Rodacki Gomes, Dr.

IFC campus Blumenau

Blumenau

2023

LISTA_ASSINATURAS Nº 323/2023 - CCCOMP/BLU (11.01.09.22)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 14/12/2023 18:07)
HYLSON VESCOVI NETTO
PROFESSOR ENS BASICO TECN TECNOLOGICO
CCCOMP/BLU (11.01.09.22)
Matrícula: ###100#1

(Assinado digitalmente em 18/12/2023 13:55)
PAULO CESAR RODACKI GOMES
PROFESSOR ENS BASICO TECN TECNOLOGICO
CGE/BLU (11.01.09.01.03.07)
Matrícula: ###299#3

(Assinado digitalmente em 15/12/2023 00:05)
RICARDO DE LA ROCHA LADEIRA
PROFESSOR ENS BASICO TECN TECNOLOGICO
CGE/BLU (11.01.09.01.03.07)
Matrícula: ###779#0

*Visualize o documento original em https://sig.ifc.edu.br/documentos/ informando seu número: **323**, ano: **2023**, tipo: **LISTA_ASSINATURAS**, data de emissão: **14/12/2023** e o código de verificação: **b26772e49e***

Agradecimentos

Agradeço a Deus por me dar forças para a realização e conclusão do curso.

À minha família que sempre me apoiou.

Aos meus professores, em especial ao professor Éder Penharbel que me orientou durante a maior parte do tempo mas infelizmente não pode estar presente no fim do processo, e ao professor Ricardo Ladeira que me orientou no final do processo e sempre contribuiu de bom grado durante todo processo de desenvolvimento.

Também aos meus colegas, em especial ao Gabriel Lima que me acompanhou durante todo o curso e também colaborou no processo de desenvolvimento deste trabalho.

Resumo

O presente trabalho de conclusão de curso apresenta a arquitetura e implementação de um protótipo de um sistema de arquivos baseado em tabela de alocação com um sistema de *tagging* integrado. É abordado o funcionamento do sistema de arquivos base, e do sistema de *tagging*, como são suas estruturas e como elas se relacionam. O resultado consiste em um programa que como um interpretador de comandos permite interagir com o sistema de arquivos que executa sobre o sistema operacional utilizando um arquivo que simula um dispositivo. O sistema desenvolvido é comparado a outros trabalhos que buscam adicionar funcionalidades de *tagging* aos arquivos com diferentes abordagens. Ele também tem seus pontos fracos analisados para poderem ser aperfeiçoados em trabalhos futuros. Por fim se tem um protótipo funcional de sistema de arquivos com *tagging* que demonstra que é possível adicionar essa nova camada de acesso a arquivos diretamente em um sistema de arquivos baseado em diretórios, com isso abre espaço para projetos futuros que ultrapassem a fase de protótipo.

Palavras-chave: Sistema de arquivos; *tagging*; Integração.

Abstract

This course completion work presents the architecture and implementation of an allocation table based file system with an integrated *tagging* system's prototype. The operation of the base file system and the tagging system are covered, as well as their structures and how they relate to each other. The result consists in a program that, like a command interpreter, allows to interact with the file system that runs on top of the operating system using a file that simulates a device. The system developed is compared to other works that seek to add tagging functionalities to files using different approaches. It also has its weaknesses analyzed so that they can be improved in future work. Finally, we have a functional prototype of a file system with tagging that demonstrates that it is possible to add this new layer of file access directly to a directories based file system, thereby opening up space for future projects that go beyond the prototype phase.

Keywords: File System; Tagging; Integration.

Lista de Ilustrações

Figura 1 - Marcadores do Gmail.....22
Figura 2 - Tabela de alocação de arquivos.....27
Figura 3 - Diagrama da estrutura do arquivo.....28
Figura 4 - Diagrama da estrutura de entrada de diretório.....29
Figura 5 - Diagrama da estrutura do sistema.....29
Figura 6 - Estrutura de arquivos.....31
Figura 7 - Estrutura de diretório.....32
Figura 8 - Estrutura da tag.....33
Figura 9 - Estrutura da lista de arquivos.....33
Figura 10 - Estrutura do sistema de *tags*.....34
Figura 11 - Visualização do *buffer* da lista de arquivos.....36
Figura 12 - Mapeamento da árvore.....37
Figura 13 - Layout plano de árvore AVL.....37
Figura 14 - Estrutura de *tags* em arquivo.....40
Figura 15 - Estrutura de *tags* em diretório.....41

Lista de Quadros

Quadro 1 - Quadro comparativo entre trabalhos.....51

Lista de Abreviaturas e Siglas

B	*Bytes*
FAT	*File Allocation Table*
GB	*Gigabyte*
HD	*Hard Drive*
MB	*Megabyte*
OSD	Object-based Storage Device
POSIX	*Portable Operating System Interface*
SA	Sistema de arquivos
SSD	Solid State Drive

Sumário

1 Introdução.....15
1.1 Justificativa.....16
1.1.1 Busca Semântica.....17
1.1.2 Limites de Sistemas Puramente Hierárquicos.....18
1.1.3 Desvantagens de Sistemas Externos.....18
1.2 Objetivos.....19
1.2.1 Objetivo Geral.....19
1.2.2 Objetivos Específicos.....19
1.3 Estrutura do Trabalho.....20
2 Fundamentação Teórica.....20
2.1 Sistema de Arquivos.....21
2.2 Tagging.....22
3 Materiais e Métodos.....23
3.1 Materiais.....23
3.1.1 Software.....23
3.1.2 Projeto e Implementação.....23
3.2 Métodos.....24
3.2.1 Revisão Bibliográfica.....24
3.2.2 Prototipação e Experimentos.....25
4 Desenvolvimento.....26
4.1 Sistema de Arquivos Base.....26
4.1.1 Arquitetura.....26
4.1.1.1 Endereçamento.....26
4.1.1.2 Arquivo.....28
4.1.1.3 Diretório.....28

4.1.1.4 Sistema.....*29*

4.1.2 Implementação.....**30**

4.1.2.1 Sistema.....*30*

4.1.2.2 Arquivos.....*30*

4.1.2.3 Diretórios.....*31*

4.2 Sistema de *Tagging*.....32

4.2.1 Arquitetura.....**32**

4.2.1.1 Tag.....*32*

4.2.1.2 Listas de Arquivos.....*33*

4.2.1.3 Árvore.....*34*

4.2.2 Implementação.....**35**

4.2.2.1 Tags.....*35*

4.2.2.2 Lista de Arquivos.....*35*

4.2.2.3 Árvore de Tags.....*36*

4.3 Integração de Sistemas.....37

4.3.1 Arquitetura.....**37**

4.3.2 Implementação.....**39**

5 Resultados e Análise.....**41**

5.1 Resultados.....41

5.1.1 Formatação e Execução.....**42**

5.1.2 Comando: ls.....**42**

5.1.3 Comando: touch.....**43**

5.1.4 Comando: echo.....**43**

5.1.5 Comando: cat.....**44**

5.1.6 Comando: mdkir.....**44**

5.1.7 Comando: rm.....**44**

5.1.8 Comando: cd.....**45**

5.1.9 Comando: namefind.....46

5.1.10 Comando: ltags.....47

5.1.11 Comando: ctags.....47

5.1.12 Comando: addtag.....47

5.1.13 Comando: rmtag.....48

5.1.14 Comando: tagfind.....48

5.1.15 Comando: find.....49

5.2 Análise.....49

5.2.1 Comparação com Trabalhos Correlatos.....49

5.2.2 Desempenho.....51

6 Conclusões e Trabalhos Futuros.....53

Referências.....55

1 Introdução

Diversas informações precisam ser armazenadas de forma permanente em sistemas computacionais, para isso, elas precisam ser gravadas na memória secundária do computador. Para uma melhor organização e padronização, essas informações são contidas em estruturas que são amplamente conhecidas, o arquivo e diretório, que permitem com que o usuário interaja com essas informações de forma conveniente.

O sistema de arquivos é a parte do sistema operacional que é responsável por fazer essa abstração e permitir que diversas operações possam ser feitas com essas informações, como salvar, editar, deletar de forma estável, ou seja, de forma com que as informações não sejam corrompidas durante nenhum processo.

Sistemas de arquivos modernos são estruturados por meio de arquivos e diretórios, com diretórios sendo estruturas que podem armazenar diversos arquivos. Essa estrutura, que pode ser classificada como hierárquica, fornece um caminho único para cada arquivo do sistema, sendo esse a cadeia de diretórios que leva até o arquivo desejado.

O tamanho médio dos dispositivos de armazenamento em computadores de uso pessoal vem crescendo cada vez mais. De acordo com Seltzer e Murphy (2009), em 1992 o espaço em disco “típico” era de aproximadamente 300MB, em 2009 esse número chegava a 300GB e atualmente é comum ver computadores pessoais com 1TB de armazenamento ou mais.

A ampliação da capacidade de armazenamento permite que quantidades cada vez maiores de arquivos possam ser armazenadas e, consequentemente, exigem uma melhora na organização e gerenciamento dos dados e uma estrutura puramente baseada em diretórios se torna um fator limitante, discutido na seção 1.1.2 (Limites de Sistemas Puramente Hierárquicos).

Uma estratégia que é utilizada para permitir uma busca em grande quantidade de informações é o *tagging*, que se baseia em relacionar palavras-chave a objetos de forma que tenham relação com seu conteúdo, assim permitindo que a busca por um objeto também seja feita através das palavras-chave associadas a ele além do seu nome. Bloehdorn *et al*. (2006) dizem que “Recentemente, *tagging* se tornou uma abordagem alternativa para organizar recursos semanticamente que tem crescido em popularidade.”

Este trabalho apresenta uma proposta de arquitetura, e um protótipo deĩ sistema de arquivos com um sistema de *tagging* integrado a fim de provar a possibilidade de dada integração.

Sistemas de arquivos são uma parte integrada do sistema operacional, por isso, para ser possível fazer um sistema totalmente funcional com chamadas de sistema adicionais é preciso fazer o desenvolvimento a nível de kernel, o que está fora do escopo deste trabalho.

Além disso, por se tratar de um protótipo que foi desenvolvido durante o estudo inicial de sistemas de arquivos, ele não faz uso de estruturas do estado da arte no contexto de sistemas de arquivos. A implementação do sistema de *tagging* tem como propósito demonstrar a possibilidade da integração com um sistema de arquivos e não apresentar uma implementação final, ou seja, existem aspectos de desenvolvimento que podem ser otimizados e são discutidos na seção 5.2.2 (Desempenho).

Durante o trabalho é comentado diversas vezes sobre a tabela de alocação do sistema, entretanto, o sistema desenvolvido não tem nenhuma relação direta com o FAT (*File Allocation Table*, que em português significa Tabela de Alocação de Arquivos), sistema de arquivos desenvolvido pela Microsoft.

1.1 Justificativa

É comum que um usuário tenha uma grande quantidade de arquivos para armazenar suas informações. A inflexibilidade de organização de arquivos em um sistema hierárquico traz o problema da localização da informação desejada - a informação certamente estará presente em um arquivo, porém é difícil determinar qual é o arquivo que contém a informação bem como, qual é o caminho do diretório que contém o arquivo (SELTZER; MURPHY, 2009). Além disso, a estrutura hierárquica restringe o acesso a um arquivo por meio de um único caminho (BLOEHDORN et al., 2006).

Sistemas de arquivos baseados em diretórios podem formar uma estrutura hierárquica, por isso daqui por diante, o uso termo “sistema hierárquico” se refere a um sistema de arquivos baseado em diretórios.

Em um sistema hierárquico, um arquivo estará localizado em um diretório, esse diretório poderá estar em outro diretório e este, por sua vez, poderá estar em uma cadeia

arbitrariamente longa de diretórios. Essa cadeia de diretórios é o caminho, embora ela possa ser arbitrariamente longa ela é única, consequentemente não existem dois caminhos diferentes para o mesmo arquivo.

Em sistemas operacionais padrão Unix é comum, para localizar um arquivo, utilizar o comando `find`, também é comum combinar o `find` com outras ferramentas como, por exemplo, o `grep` - um filtro de *string* por expressões regulares. Essa busca é efetiva em termos de nomes de arquivos, ou seja, em termos de encontrar o nome do arquivo ou não.

Porém, ao localizar um arquivo por seu nome, surgem alguns problemas de ordem prática, entre eles: (1) nem sempre o usuário se recorda do nome do arquivo, (2) o conteúdo do arquivo é desprezado durante a busca pelo nome do arquivo e (3) é uma busca textual que concatena todos os nomes de arquivos um após o outro, assim, consequentemente, essa busca não aproveita de estruturas de dados mais avançadas como, por exemplo, árvores de busca.

É possível buscar por expressões regulares dentro de arquivos com, por exemplo, o `grep`, entretanto essa é uma operação que também não faz uso de estruturas de dados que poderiam melhorar o desempenho em termos de tempo de execução.

1.1.1 Busca Semântica

Buscas com base em conteúdo, ou seja, com base semântica são comuns no dia a dia, elas estão presentes nos principais motores de busca da internet como Google, Bing e DuckDuckGo. Embora esse tipo de busca seja muito comum para informações na internet, elas não estão presentes em sistemas de arquivos tradicionais, Gifford et al. (1991) sugerem que sistemas de arquivos semânticos proporcionam melhores resultados para buscas.

Classificar a informação com *tags* é uma maneira de qualificar a informação para facilitar a sua busca. Elas auxiliam a organização semântica da informação, podem ser facilmente gerenciáveis e, portanto, passíveis de serem incorporadas às estruturas internas do sistema. É um mecanismo que amplia o potencial de localização da informação.

1.1.2 Limites de Sistemas Puramente Hierárquicos

Sistemas de arquivos tradicionais são hierárquicos, ou seja, internamente formam uma estrutura de árvore. Os diretórios formam os nós internos e os arquivos são os nós externos ou as folhas.

Uma estrutura puramente hierárquica traz limitações e, por mais que se tente, através dos nomes dos arquivos e diretórios, determinar uma relação semântica entre os arquivos e a informação contida neles, é fácil acabar criando situações que se tornam não lógicas.

Um exemplo é mostrado em (BLOEHDORN et al., 2006) no qual facilmente é possível criar relações redundantes como os “computação/artigos” e “artigos/computação”.

Outro exemplo de situação que esbarra na unicidade de caminhos é quando um usuário deseja armazenar um livro de algum assunto duplo como, por exemplo, arte e arquitetura. Se o usuário possuir diretórios diferentes para assuntos de arte e de arquitetura, então será necessário decidir em qual dos diretórios será armazenado o conteúdo.

Não existe uma solução 100% efetiva, caso ele coloque em somente uma das pastas suas pastas não estarão em total acordo com o conteúdo do livro, caso ele coloque nas duas pastas, terá arquivos duplicados ocupando espaço desnecessário em seu armazenamento.

Outro possível recurso seria fazer um link simbólico do arquivo para outra pasta, mas esse é um processo que adiciona complexidade desnecessária e que pode trazer problemas futuros como, por exemplo, durante o processo de backup.

O uso de *tags* resolve essa situação, é possível colocar o arquivo em uma pasta com nome mais abrangente como “Livros” e relacionar os conteúdos presentes nele por meio de uma tag. A partir da *tag* será possível organizá-lo e encontrá-lo sem a necessidade de duplicar os arquivos ou de criar links.

1.1.3 Desvantagens de Sistemas Externos

É possível recorrer a aplicativos que oferecem um sistema de *tagging*, entretanto, aplicativos não proporcionam as vantagens de estar diretamente conectados ao sistema de arquivos.

Um exemplo de aplicação que oferece essa funcionalidade é o TagSpaces[1], ele permite organizar os arquivos do usuário com *tags*, ele faz essas relações por meio de novos arquivos que armazenam as relações criadas ou então alterado o nome dos arquivos para adicionar as informações de *tags* concatenando as *strings* das *tags* ao nome original, dessa forma a estrutura de *tag* não consegue utilizar estruturas mais avançadas para busca e no caso do armazenamento das relações em arquivos, ficam sujeitas a fácil perda de integridade por todas as estruturas estarem diretamente abertas ao usuário além de ficar diretamente dependente da aplicação para seu funcionamento.

Outro aplicativo que oferece essa funcionalidade é o TMSU[2] que inicializa um banco de dados próprio para guardar as relações de *tags* com arquivos, dessa forma as *tags* estão ligadas somente à aplicação e outras operações com arquivos podem comprometer a integridade do sistema, outras estratégias são apresentadas na seção 3.2.1 (Revisão Bibliográfica) e discutidas na seção 5.2.1 (Comparação com Trabalhos Correlatos).

A principal vantagem de um sistema de *tagging* integrado a um sistema de arquivos é que ele mantém a integridade por poder se comunicar diretamente com as outras funções do sistema, além de poder oferecer a funcionalidade mediante uma interface para outros programas de forma padronizada, assim tornando o funcionamento das *tags* independente de programas específicos.

1.2 Objetivos

Nesta seção é apresentado o objetivo geral seguido dos objetivos específicos do trabalho.

1.2.1 Objetivo Geral

Implementar um protótipo de sistema de *tagging* integrado a um sistema de arquivos.

1.2.2 Objetivos Específicos

1 Mais informações em: https://www.tagspaces.org/

2 Mais informações em: https://tmsu.org/

a) Desenvolver um sistema de arquivos baseado em tabela de alocação;
b) Elaborar uma arquitetura de *tagging* interna e integrada ao sistema de arquivos.

1.3 Estrutura do Trabalho

Os tópicos abordados apresentados neste trabalho se dividem da seguinte forma:

- Na seção 1 (Introdução) são apresentados conceitos básicos que serão abordados no trabalho, os objetivos e a justificativa do trabalho;
- Na seção 2 (Fundamentação Teórica) são apresentados com mais detalhes os conceitos-chave envolvidos no trabalho, que são: Sistema de arquivos e sistema de *tagging*;
- Na seção 3 (Materiais e Métodos) é apresentada uma revisão bibliográfica, os métodos utilizados no trabalho, os recursos de software utilizados no desenvolvimento;
- Na seção 4 (Desenvolvimento) são apresentados os aspectos relacionados à arquitetura e ao desenvolvimento das três principais partes do trabalho: o sistema de arquivo base, o sistema de *tagging* e a integração dos dois sistemas.
- Na seção 5 (Resultados e Análise) é apresentado o sistema de arquivos base integrado ao sistema de *tagging* funcionando, são explicados e apresentados exemplos das principais funcionalidades do sistema final. Também é feita uma comparação entre trabalhos correlatos e o sistema apresentado nesse trabalho e uma análise com relação ao desempenho do sistema;
- Na seção 6 (Conclusões e Trabalhos Futuros) são apresentadas considerações finais sobre o trabalho e trabalhos futuros que abordam melhorias ao trabalho atual e a adição de novas funcionalidades.

2 Fundamentação Teórica

Nesta seção são apresentados os conceitos básicos sobre os temas abordados para que seja possível ter um entendimento mais claro do trabalho, sendo eles: Sistemas de arquivos e *tagging.*

2.1 Sistema de Arquivos

Dispositivos de armazenamento como HDs (*Hard Drive*), SSDs (*Solid State Drive*) e *pen drives* são dispositivos de memória não volátil e basicamente grandes espaços de memórias onde bits podem ser gravados e lidos. O sistema de arquivos organiza esse espaço de memória formando uma interface entre o usuário e os dados contidos nesse espaço.

A estrutura atômica que prove essa organização é o arquivo, “O sistema operacional abstrai das propriedades físicas do seu dispositivo para definir uma unidade de armazenamento lógica, o arquivo” (SILBERSCHATZ; GALVIN; GAGNE, 2018).

De acordo com Tanenbaum (2015) todo arquivo possui dados e um nome, além de outros atributos que podem variar de sistema para sistema. Arquivos costumam ser utilizados juntamente com estruturas de diretórios, que também são arquivos, mas que armazenam referências para outros arquivos/diretórios, permitindo assim uma estrutura de árvore.

O sistema de arquivos é capaz de permitir a criação, gerenciamento e manipulação dessas estruturas que são adotadas como padrão.

Para alcançar essa organização o sistema de arquivos precisa fornecer uma maneira de acessar todos os espaços de memória, não é possível realizar esse acesso de forma direta a cada *byte* do dispositivo.

Em dispositivos com grande capacidade de armazenamento como, por exemplo, discos rígidos, é necessário considerar uma representação funcional desses endereços. Uma unidade de 5GB possui mais *bytes* do que o máximo que um inteiro de 32 bits pode armazenar (5GB equivalem a 5.000.000.000 *bytes*, *unsigned long* tem sua faixa de valores de 0 a 4.294.967.295).

Atualmente são comuns computadores com dispositivos que armazenam até 1TB, portanto, o sistema de arquivos precisa prover uma forma de endereçamento para que todos os espaços de memória possam ser acessados.

O endereçamento costuma ser feito a partir da divisão do dispositivo físico em blocos lógicos, permitindo com que trechos dos arquivos sejam mapeados para esses blocos, “[…] o arquivo pode ser considerado uma sequência de blocos. Toda operação básica de E/S opera em termos de blocos.” (SILBERSCHATZ; GALVIN; GAGNE, 2018).

Com um mapeamento baseado em blocos temos uma forma de endereçamento indireto, obtendo o endereço de um bloco do sistema é possível acessa um novo endereço que está contido dentro deste bloco tornando possível mapear quantias de memória muito maiores.

É tarefa do sistema de arquivos definir quais são os espaços utilizados e quais estão disponíveis para a escrita.

2.2 Tagging

Um sistema de *tagging* se baseia na marcação ou associação de palavras-chave a objetos com o intuito de permitir uma maior organização. No contexto de sistema de arquivos, com a aplicação das *tags* aos arquivos é possível classificar, organizar e encontrar arquivos de maneira mais rápida e prática.

O principal objetivo de uma *tag* é associar o arquivo a uma cadeia de caracteres que expresse algum sentido relacionado ao conteúdo, ou seja, que haja uma relação semântica.

Depois que um arquivo se relacionar a uma tag, é possível encontrá-lo fazendo uma busca pela tag.

Recursos marcados com *tags* formam implicitamente conjuntos, cada conjunto está relacionado a uma palavra-chave, dessa forma é possível fazer operações de conjunto como união, intersecção e diferença, permitindo um comportamento semelhante a um banco de dados.

Um exemplo clássico do uso de um sistema de *tags* são os marcadores do Gmail (Figura 1).

Figura 1 - Marcadores do Gmail

Fonte: NetBit (2018)

3 Materiais e Métodos

Nesta seção são apresentados os materiais e métodos utilizados para o desenvolvimento do trabalho.

3.1 Materiais

Nesta seção são apresentados os materiais usados para o desenvolvimento do trabalho, apresentando o ambiente no qual o sistema apresentado foi desenvolvido, as bibliotecas utilizadas para o seu desenvolvimento e a maneira como ele foi implementado.

3.1.1 Software

O sistema foi desenvolvido e testado utilizando Linux com kernel versão 6.5.7. Esse sistema pode ser compilado tanto para Linux como para Windows utilizando o compilador gcc versão 13.2.1 e utilizando as bibliotecas math.h e *pthread*. Não é necessário ser um usuário administrador para utilizar o sistema.

3.1.2 Projeto e Implementação

Para o desenvolvimento do sistema base foi estudada a arquitetura geral de um sistema de arquivos apresentada em (TANENBAUM, 2015). Foi utilizada a linguagem C utilizando somente bibliotecas padrões para a manipulação de memória, entrada e saída:

- stdlib.h: Utilizada para acesso a funções de manipulação de memória dinâmica, como `malloc`, `realloc` e `free`;
- stdio.h: Utilizado para acesso a funções de entrada e saída, como `printf` e `fgets`;
- string.h: Utilizado para acesso a funções de manipulação de *strings* e memória, como `strcpy`, `strcmp`, `strcat`, `memcpy`;
- stdint.h: Utilizado para ter acesso a macros e utilidades relacionadas a inteiros como uint e `__INT16_MAX__`;

- windows.h, local.h: Utilizado para acesso a funções que permitem a compatibilidade da interface com Windows;
- unistd.h: Utilizado para acesso a funções que permitem a compatibilidade da interface com Linux;
- time.h: Utilizado para acesso a funções de tempo.

O sistema de arquivos desenvolvido é um programa comum que recebe como argumento um arquivo que simula um dispositivo de armazenamento, esse programa é capaz de retornar trechos do arquivo como se fossem os blocos de um dispositivo de armazenamento. As operações são realizadas nesse arquivo como se ele fosse o dispositivo com o sistema "montado".

É utilizado o comando `truncate` no Linux para criar um arquivo do tamanho desejado. O arquivo gerado pelo `truncate` representa o dispositivo de memória secundária e é manipulado através das chamadas de sistema `read` e `write`.

Para integrar o sistema proposto em algum *kernel* é preciso adequar as chamadas de leitura e escrita para escrever diretamente no dispositivo e a interface POSIX (*Portable Operating System Interface*).

3.2 Métodos

Nesta seção são apresentados os métodos utilizados para o desenvolvimento deste trabalho, sendo eles o experimental, e o bibliográfico através da revisão bibliográfica, que também apresentará trabalhos correlatos.

3.2.1 Revisão Bibliográfica

O conjunto das referências bibliográficas são fruto de um estudo da literatura para que fossem identificadas bibliografias pertinentes ao assunto. Foram escolhidos textos para o direcionamento do desenvolvimento e, consequentemente, para a sustentação da justificativa da proposta.

Existem trabalhos que propõem a adição de relações semânticas a sistemas de arquivos, mas possuem abordagens diferentes e com propostas finais diferentes. Na seção 5.2.1 (Comparação com Trabalhos Correlatos) os produtos dos trabalhos apresentados a seguir são comparados com o sistema apresentado neste trabalho.

Bloehdorn *et al*. (2006) apresentam o TagFS, um sistema de arquivos baseado no uso de *tags* com sua implementação baseada em WebDAV, é uma interface sobre o sistema de arquivos dos servidores que hospedam o WebDAV. As *tags* no sistema são geradas de forma implícita através da operação de envio do arquivo para dentro do servidor WebDAV, o caminho completo do arquivo é utilizado para gerar uma abstração de *tags*, fica, portanto impedida a adição e remoção explícita de *tags*.

Gifford et al. (1991) apresentam o SFS (Semantic File System) que também é uma implementação de uma interface sobre o sistema de arquivos, para atingir o objetivo de um sistema de arquivos semântico não são utilizadas exatamente *tags*, mas sim conjuntos de campos com valores específicos para cada tipo de arquivos. São empregados programas personalizáveis para executar a análise e preenchimento dos campos de valores que servem como *metadata* e geram a estrutura de diretórios virtuais.

Seltzer e Murphy (2009) apresentam uma arquitetura para um sistema de arquivos semânticos, o hFAD (Hierarchical File Systems are Dead), na qual é removido totalmente a estrutura hierárquica, a estrutura de diretórios é usada somente para compatibilidade com o POSIX, a arquitetura consiste em uma estrutura semelhante a um OSD (Object-based Storage Device) com conteúdos dos arquivos armazenados em objetos, que possuem identificadores únicos e metadados.

3.2.2 Prototipação e Experimentos

Para comprovar o funcionamento da integração dos dois sistemas foi implementada uma interface que funciona como um interpretador de comandos para que seja possível utilizar as funcionalidade do sistema de arquivos integrado ao sistema de tagging.

A comprovação do funcionamento se deu pela execução desses comandos, permitindo com que fossem testadas e validadas todas as funcionalidades do sistema

desenvolvido. A apresentação dessa interface assim como os resultados dos testes feitos são apresentados na seção 5.1 (Resultados).

4 Desenvolvimento

Nesta seção serão apresentados aspectos de arquitetura e implementação dos principais componentes desse trabalho, o sistema de arquivo base, o sistema de *tagging* e integração dos dois sistemas.

4.1 Sistema de Arquivos Base

Um sistema de arquivos foi implementado para ser possível realizar a integração de um sistema de *tagging* sem que fosse necessário lidar com os sistemas de arquivos já em uso a nível de kernel nos sistemas operacionais correntes como, por exemplo, Linux ou FreeBSD. Isso traria uma complexidade que não se enquadra ao escopo do trabalho.

Nesta seção serão abordados os conceitos de arquitetura e implementação utilizados no desenvolvimento do sistema.

4.1.1 Arquitetura

Existem diferentes formas de criar as partes de um sistema de arquivos, por isso que existem diversos sistemas distintos, nesta seção é abordada a arquitetura utilizada nos principais componentes do sistema.

4.1.1.1 Endereçamento

O endereçamento dos espaços de memória do dispositivo é feito por meio de uma tabela de alocação de arquivos, por ser uma estrutura mais simples, levando em consideração que o sistema foi construído durante o estudo inicial de sistema arquivos.

No processo de formatação é possível definir o tamanho do bloco do sistema, a partir desse tamanho é feita a divisão lógica de todo o dispositivo em blocos e então é gerada uma tabela de tamanho *X* com *X* sendo a quantidade de blocos que formam o dispositivo.

Na tabela, cada índice representa um bloco do dispositivo e está associado um valor *N* tal que *N* pertence ao conjunto dos números inteiros de forma que $-1 \leq N < M$ com *M* sendo a quantidade de blocos que o dispositivo foi dividido.

Quando *N* é igual a -1 indica que o bloco está ocupado e não existe um próximo bloco com informações para ser visitado.

Quando *N* é igual a 0 indica que o bloco está disponível para a escrita.

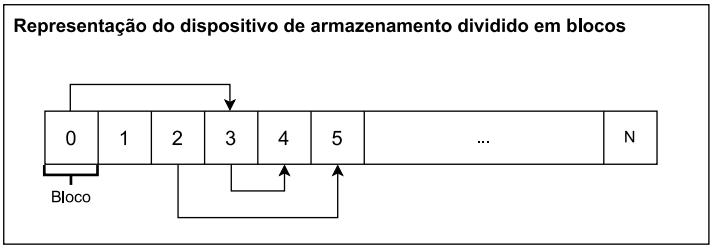
Caso *N* seja maior que 0 o valor de *N* indica o índice do próximo bloco que deve ser acessado para continuar a leituras dos dados da estrutura que está contida naquele bloco, ou seja, formando uma estrutura de lista encadeada.

Este modelo de endereçamento é descrito por Tanenbaum (2015) como “Alocação com lista encadeada utilizado tabela em memória”.

Na Figura 2 é possível ver uma representação da tabela gerada e do dispositivo dividido em blocos, assim como a relação entre os blocos que permite com que estruturas sejam armazenadas em mais de um bloco.

Figura 2 - Tabela de alocação de arquivos.

Tabela de alocação	
Índice	Valor
0	3
1	0
2	5
3	4
4	-1
5	-1
...	...
N	0

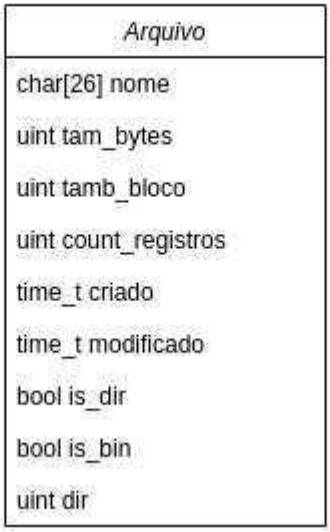


Fonte: Elaborado pelo autor (2023)

4.1.1.2 Arquivo

No sistema, um arquivo é composto por um cabeçalho onde são armazenados os seus metadados, e seu conteúdo. As informações que o arquivo armazena em seu cabeçalho são apresentadas na Figura 3.

Figura 3 - Diagrama da estrutura do arquivo



Fonte: Elaborado pelo autor (2023)

Por questão de simplicidade de implementação, o identificador do arquivo é o número do bloco inicial em que ele está armazenado. No processo de criação do arquivo é indicado o índice do primeiro bloco que irá armazenar suas informações, é então reservado uma quantidade de blocos que serão necessários para armazenar todo o conteúdo do arquivo.

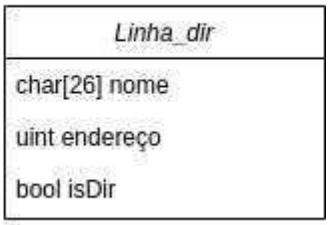
Esses blocos são ligados por meio de uma lista encadeada na tabela de alocação com o índice na tabela contendo a informação que indica o próximo índice a ser consultado para escrever/ler as informações do arquivo, com o índice do último desses blocos marcado com -1 para identificar o último bloco reservado para o conteúdo do arquivo, assim formando uma estrutura conforme a Figura 2.

4.1.1.3 Diretório

Um diretório possui as mesmas informações e maneira de endereçamento do arquivo, entretanto o conteúdo que ele armazena é uma sequência de estruturas que definem entradas

de diretório, que fazem referência a arquivos, essas entradas armazenam as seguintes informações mostradas na Figura 4 :

Figura 4 - Diagrama da estrutura de entrada de diretório



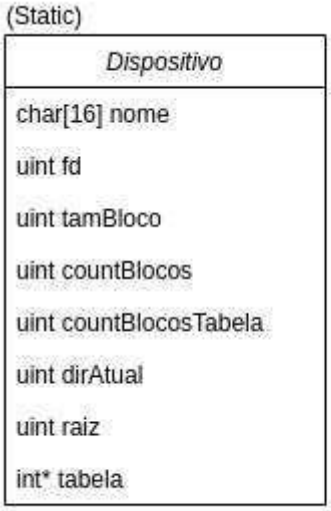
Fonte: Elaborado pelo autor (2023)

4.1.1.4 Sistema

O sistema apresentado possui uma estrutura simples, a estrutura consiste em um espaço reservado no início do dispositivo onde são armazenadas informações referentes ao dispositivo e a tabela alocação de arquivos, esse espaço é reservado durante o processo de formatação do dispositivo, momento em que é possível identificar o tamanho desejado para cada bloco.

Informações contidas na estrutura do sistema no espaço reservado são mostradas na Figura 5:

Figura 5 - Diagrama da estrutura do sistema



Fonte: Elaborado pelo autor (2023)

4.1.2 Implementação

A implementação é a arquitetura na prática. Essa seção apresenta questões relacionadas à implementação das principais partes do sistema, ou seja, como cada parte se comporta no software.

4.1.2.1 Sistema

O sistema possui uma estrutura global que pode ser acessada por todas as funções que compõem o sistema de arquivos, justamente por ela armazenar as informações essenciais para a operação do sistema, listadas na seção 4.1.1.4 (Sistema). Essas informações, assim como a própria tabela de alocação ficam armazenadas na memória durante o funcionamento do sistema e toda vez modificadas também têm seus estados gravados nos blocos reservados para elas.

A estrutura do sistema, assim como a tabela de alocação, é gravada no início do dispositivo. Os dados são criados quando o dispositivo é formatado, recuperados sempre que o sistema é executado em um dispositivo já formatado e gravados no disco sempre que são modificados.

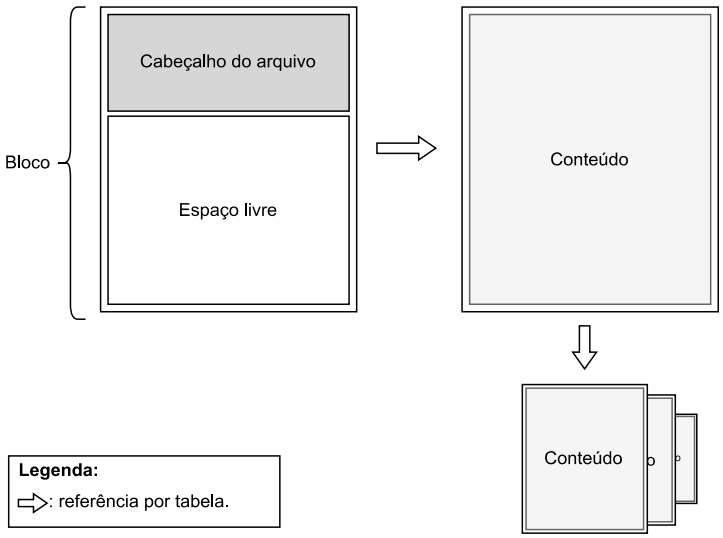
4.1.2.2 Arquivos

Um arquivo possui seu cabeçalho e seu conteúdo. No sistema, o identificador do arquivo, por questões de simplicidade de implementação, é o mesmo índice do bloco em que seu cabeçalho está gravado, ou seja, o índice do bloco que armazena as primeiras informações do arquivo. No bloco inicial do arquivo está presente o cabeçalho do arquivo, que possui um tamanho fixo.

O conteúdo do arquivo não está presente no seu bloco inicial, mas sim em um próximo bloco indicado pelo valor no índice do bloco inicial na tabela de alocação, que indica outro bloco que pode indicar outro bloco ou então sinalizar o fim do arquivo conforme

explicado na seção 4.1.1.1 (Endereçamento). A Figura 6 apresenta uma visualização do funcionamento da estrutura do arquivo.

Figura 6 - Estrutura de arquivos

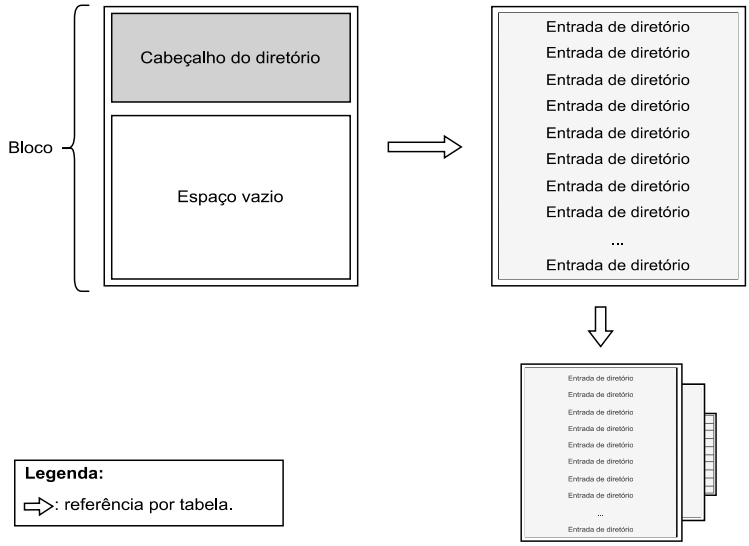


Fonte: Elaborado pelo autor (2023)

4.1.2.3 Diretórios

Diretórios possuem a mesma estrutura apresentada no arquivo, entretanto seu conteúdo é composto por entrada de diretórios ao invés de informações representadas em binário como em um arquivo regular. A Figura 7 apresenta uma visualização do funcionamento da estrutura do diretório.

Figura 7 - Estrutura de diretório.



Fonte: Elaborado pelo autor (2023)

4.2 Sistema de *Tagging*

O sistema de *tagging* é o sistema responsável pela relação de palavras-chave com os arquivos, sua finalidade é criar a relação semântica entre os arquivos e um marcador.

4.2.1 Arquitetura

Existem várias formas de criar sistemas de *tagging*, algumas dessas formas são apresentadas na seção 3.2.1 (Revisão Bibliográfica). Aqui será apresentada a arquitetura que permite que o sistema de *tagging* possa ser integrado ao sistema de arquivos.

4.2.1.1 Tag

A estrutura de *tag* é a principal estrutura do sistema, ela que permite a relação entre uma palavra-chave e um conjunto de arquivos. A estrutura de *tag* é formada por uma palavra-chave e por uma lista dos arquivos que estão associadas a ela.

O conceito de *tags* é baseado na relação do conteúdo da informação com palavras-chave, é possível utilizar múltiplas *tags* para relacionar um conteúdo com diversas palavras-chave, é possível visualizar esta estrutura na Figura 8.

Figura 8 - Estrutura da tag

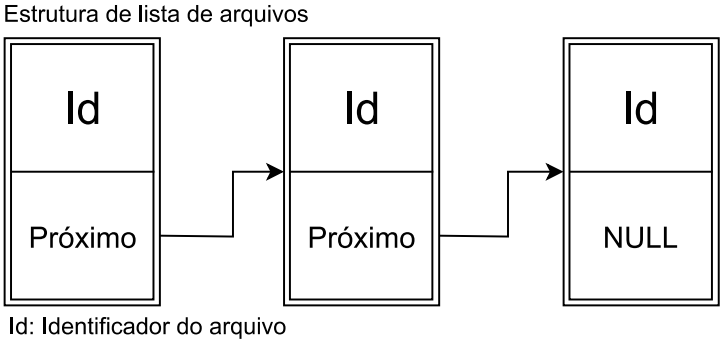
Fonte: Elaborado pelo autor (2023)

4.2.1.2 Listas de Arquivos

Para ser possível identificar quais arquivos estão relacionados com cada tag, a estrutura de *tag* deve possuir acesso a todos os arquivos que estão relacionados a ela. Uma das estratégias possíveis para estabelecer essa relação, e a utilizada neste trabalho, é por meio de uma lista encadeada contendo os identificadores dos arquivos.

Cada *tag* deve apontar para sua lista exclusiva de arquivos relacionados, dessa forma, no momento em que a *tag* é obtida, também já é possível obter diretamente os arquivos relacionados a ela. A estrutura da lista encadeada proporciona facilidade de implementação por ser uma estrutura facilmente serializável e redimensionável, é possível visualizar esta estrutura na Figura 9.

Figura 9 - Estrutura da lista de arquivos



Fonte: Elaborado pelo autor (2023)

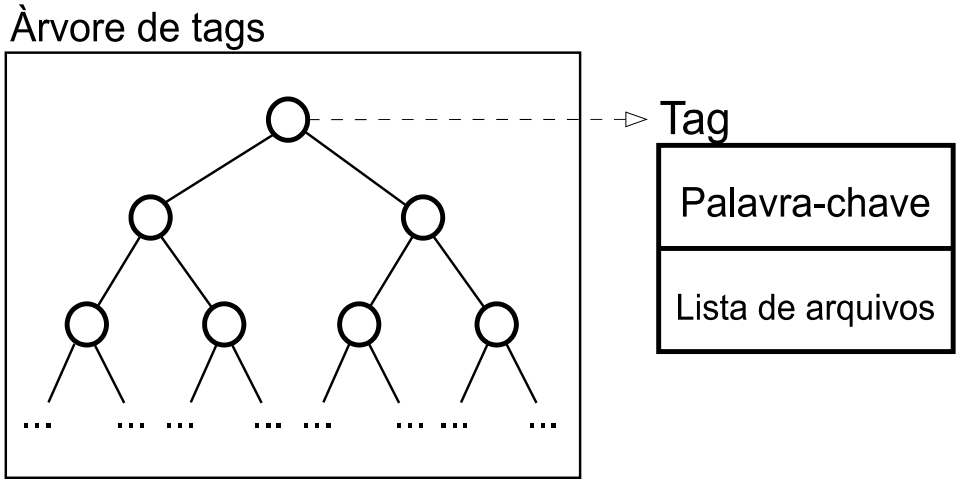
4.2.1.3 Árvore

O objetivo de um sistema de *tags* é proporcionar uma busca mais rápida aos arquivos que estão relacionados a ela, por isso, as estruturas de *tags* são armazenadas em uma árvore AVL. A árvore AVL é uma árvore binária ordenada e balanceada.

Por ser binária, cada nó da árvore possui, no máximo, dois filhos. Por ser ordenada, os filhos possuem relação de ordem com o nó pai, os filhos à esquerda serão sempre menores que o pai e os filhos a direita sempre maiores. Finalmente, por ser balanceada, é imposta a restrição de que os filhos terão diferença de altura de, no máximo, um nível, isso significa que toda vez que os filhos de um nó diferem em altura superior a dois são executadas operações de rotação para reorganizar a árvore, dessa forma, o tempo de busca em uma árvore AVL é preservado.

A árvore AVL possui uma complexidade de tempo de *O(log N)* no pior caso para busca, inserção e exclusão, onde *N* é o número de nós na árvore (STANDISH, 1980), isso permite com que por mais a quantidade de *tags* aumente, que são os nós da árvore, a busca mantenha um bom desempenho, é possível visualizar a estrutura do sistema de *tagging* na Figura 10.

Figura 10 - Estrutura do sistema de *tags*



Fonte: Elaborado pelo autor (2023)

4.2.2 Implementação

A implementação é a arquitetura na prática. Essa seção apresenta questões relacionadas à implementação das partes do sistema de *tagging*, especialmente referente à serialização da estrutura, que gera dois *buffers* encapsulados, um *buffer* contendo as informações dos arquivos relacionados às *tags*, e outro contendo a estrutura da árvore de tag.

4.2.2.1 Tags

Por mais importante que seja, a implementação da estrutura de *tag* é simples, ela armazena uma *string* de tamanho fixo que pode ser delimitada no código do programa, essa limitação diminui a complexidade da implementação.

Além disso, ela também possui um ponteiro que aponta para o primeiro item de uma lista encadeada de identificadores de arquivos, que definem que os arquivos estão relacionados à tag.

4.2.2.2 Lista de Arquivos

Como cada *tag* tem uma referência para uma lista de arquivos, existe uma lista de arquivos para cada tag, com cada uma dessas listas tendo um tamanho variável. Para ser possível fazer a serialização e escrita no disco dessas listas é preciso ter a quantidade de listas e o tamanho de cada lista.

Com isso é possível organizar todos os itens dessas listas, que são os identificadores dos arquivos, de forma linear, e então fazer o mapeamento de quais itens pertencem à lista de cada uma das *tags*.

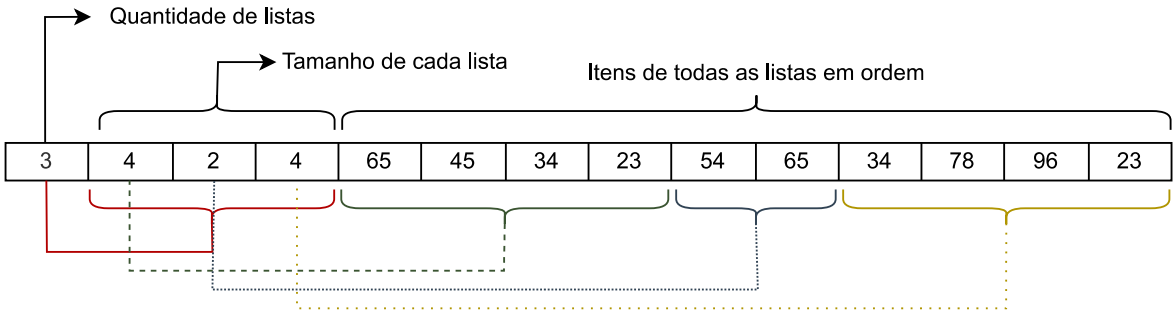
Portanto, ao transformar essas listas em um *buffer* são guardados as seguintes informações em ordem, conforme pode ser visto na Figura 11:

1. Quantidade de listas: define a quantidade de leituras de listas que precisam ser executadas;

2. Lista com as quantidades de itens em cada tag: lista que possui o tamanho da quantidade de *tags* e armazena o tamanho de cada uma das listas de cada *tag*, para poder ser feito o número correto de leituras posteriormente;

3. Itens das listas: todos os itens de todas as listas em ordem, assim com as informações anteriores é possível fazer a leitura de todas as listas e realizar a associação com cada *tag*.

Figura 11 - Visualização do *buffer* da lista de arquivos



Fonte: Elaborado pelo autor (2023)

4.2.2.3 Árvore de Tags

A árvore de *tags* permite com que seja feita uma busca eficiente dentro do sistema de *tags*, essa estrutura precisa ser escrita em disco para que não seja perdida a cada execução do programa. Para serializar e então escrever em disco essa estrutura sem que as relações entre os nós se percam, é utilizada a técnica apresentada por Pigeon (2009).

Essa solução consiste em armazenar os nós da árvore (Figura 12) em uma *array (Figura 13)* de forma que exista uma relação matemática entre os nós, tornando possível preservar a ordem da estrutura de árvore, assim permitindo uma leitura direta, sem que todos os nós precisem ser adicionados novamente na estrutura em memória e todas as rotações resultantes das inserções serem processadas.

A relação entres os nós é mantida da seguinte forma, adaptado de (PIGEON, 2009):

Para cada nó *N*,

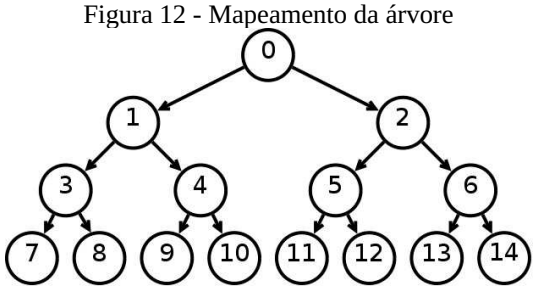
Seu pai é dado por [½ (*N* - 1)],

Seu filho da esquerda é dado por $2N+1$,

Seu filho da direita é dado por $2N+2$,

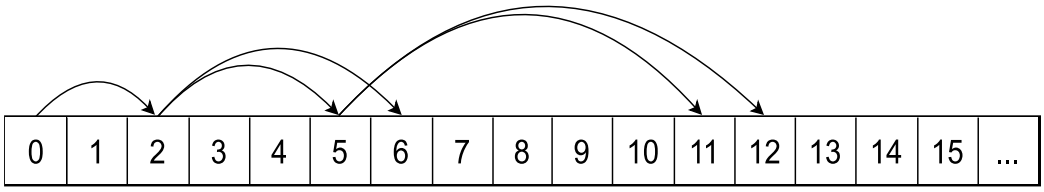
Assumindo que o nó raiz seja numerado com 0 temos o seguinte layout representado na Figura 13, ondes as flechas correspondem as ligações entre pais e filhos apresentados na Figura 12:

Figura 12 - Mapeamento da árvore



Fonte: Pigeon (2009)

Figura 13 - Layout plano de árvore AVL



4.3 Integração de Sist Fonte: Adaptado de Pigeon (2009)

O sistema de arquivos base não foi criado com o objetivo de integrar um sistema de arquivos no futuro, já o sistema de *tagging*, por mais que seja possível integrá-lo a outros tipos de sistema, ele foi planejado para que fosse possível integrá-lo a um sistema de arquivos, com os identificadores dos arquivos sendo o elo de ligação.

4.3.1 Arquitetura

A implementação do sistema de *tagging* já permite transformar toda a estrutura em *buffers*, com informações de como mapeá-los de volta para a estrutura padrão, já pensado em como essas informações teriam que ser gravadas diretamente em disco na integração com o sistema de arquivo.

Por ser uma estrutura que pode crescer indefinidamente, já que não existe um limite para a quantidade de *tags* e relações que podem ser adicionadas, não é possível alocar um espaço pré-determinado para que ela seja organizada.

Para que essa estrutura possa crescer (e diminuir) dinamicamente no disco ela é armazenada de forma semelhante a como arquivos são armazenados, por possuírem a mesma característica de ter tamanho variável. Os *buffers* que guardam as informações do sistema de *tagging* também podem ser acessados por meio de um sistema de lista encadeada no qual os *buffers*, a mediada que necessário, são divididos em blocos, e cada bloco de *buffer* possui uma referência para o próximo bloco a conter o *buffer*.

Com a estrutura de árvores de *tag* sendo guardada e recuperada do disco já é possível ter toda a funcionalidade do sistema de *tags*, porém, uma situação comum em um sistema de *tags* é a verificação de quais *tags* estão associadas a determinado objeto, no caso, ao arquivo.

Somente com a estrutura de árvore de *tags* para obter essa informação, é preciso percorrer todos os nós da árvore para verificar com quais *tags* determinado arquivo está relacionado, porém a medida que a árvore cresce o custo dessa operação por ter complexidade de tempo *O(N)* com *N* sendo a quantidade de *tags* presentes no sistema.

Para contornar essa situação, as palavras-chave das *tags* associadas com cada arquivos também estão armazenadas juntamente com o arquivo, isso permite com que seja possível acessar rapidamente todas as *tags* associadas a cada arquivo.

Como a quantidade de *tags* associadas a um arquivo também pode aumentar indefinidamente, essas palavras-chave também são armazenadas em seus blocos especiais em uma estrutura de lista encadeada, assim como funciona o armazenamento dos *buffers* da árvore de *tags*.

Com a relação de palavras-chave e arquivos sendo mantidas tanto na árvore de *tags*, como em cada arquivo, ocorre que caso um dos locais seja corrompido, será possível fazer a recuperação da informação com os dados do outro local, permitindo um grau a mais de integridade ao sistema.

4.3.2 Implementação

Como visto na seção 4.3.1 (Arquitetura), a estrutura de *tags* em árvore é convertida para *buffers* que são escritos no disco, de forma semelhante ao arquivo, formando uma estrutura de lista encadeada.

A estrutura de árvore é dividida em dois *buffers*, um que guarda somente a estrutura da árvore, ou seja, os nós, de forma que preserve seus respectivos posicionamentos, com as palavras-chave de cada nó, e outro que é responsável por armazenar as listas dos arquivos relacionados a cada uma das *tags*.

Para acomodar essas duas estruturas, é preciso adicionar os índices dos blocos que contém esses *buffers* à estrutura do sistema para que ele possa então acessar essas informações. Portanto, é adicionado a estrutura do sistema o índice do bloco que possui as informações referentes a estrutura de *tags*, sendo essas:

- Índice do bloco inicial que contém a estrutura da árvore;
- Índice do bloco inicial que contém as estruturas das listas.
- Quantidade de *tags*.
- Tamanho do *buffer* da árvore.
- Tamanho do *buffer* das listas.

Para formar a estrutura de lista encadeada, cada bloco do conjunto de blocos que armazenam o *buffer* da árvore também armazena uma estrutura que contém as informações do próximo bloco que armazena o *buffer*.

O armazenamento da estrutura de listas ocorre de forma semelhante, entretanto, como comentado na seção 4.3.1 (Arquitetura), é preciso armazenar mais do que os valores contidos nas listas para conseguir fazer o mapeamento correto das listas às *tags* armazenadas na árvore.

Os blocos que armazenam esses *buffers* de listas antes de armazenar os valores das listas também armazenam a quantidade total de listas e a quantidade de identificadores presentes em cada uma das listas, assim permitindo o mapeamento correto.

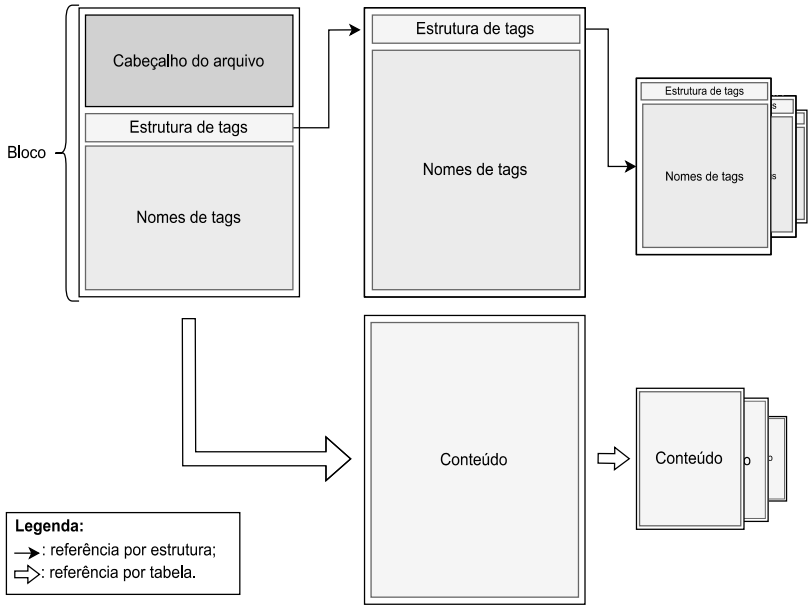
Os blocos da lista também possuem uma seção que indica o próximo bloco que deve ser consultado, para gerar a estrutura de lista encadeada.

Para o armazenamento das palavras-chave de forma que estivessem diretamente relacionadas ao arquivo foram criadas novas estruturas armazenadas diretamente após o cabeçalho do arquivo no seu bloco inicial, essa estrutura contendo informações referentes às palavras-chave que estão relacionadas ao arquivo, palavras-chave armazenadas em seguida, devido à implementação do sistema de arquivos base, essas palavras-chave podem ser diretamente salves no bloco inicial por ele possuir um espaço vago, visto que o conteúdo no arquivo nunca é escrito diretamente no bloco inicial dele.

Entretanto, a estrutura com as informações das palavras-chave, traz dentro de si a quantidade de palavras e armazenadas no bloco atual, e um índice que aponta para o próximo bloco que armazena mais *tags*, caso seja necessário, sendo assim, nada impede que o conteúdo também fosse escrito diretamente no bloco inicial, após o cabeçalho e estrutura de tag. As palavras-chave (ou nomes das *tags*) são simples *strings* gravadas em sequência.

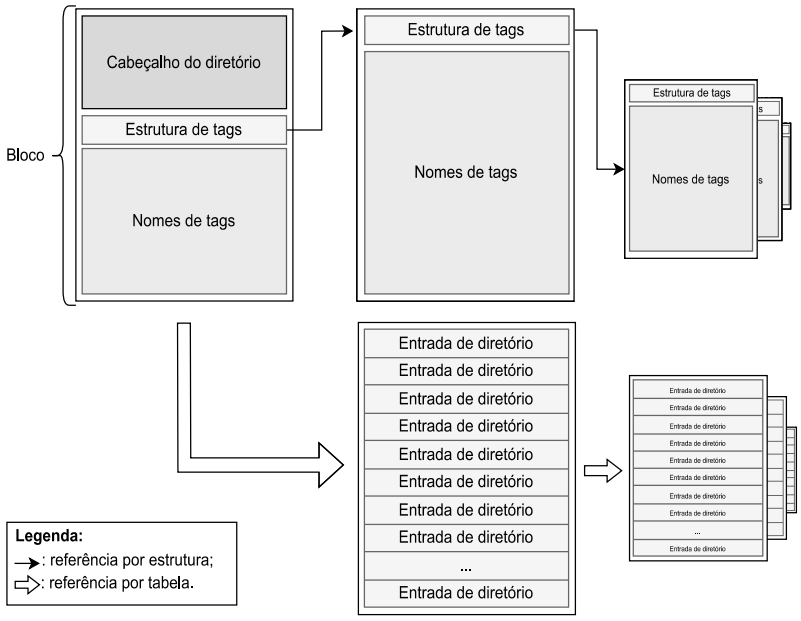
A Figura 14 apresenta uma visualização do funcionamento desse sistema, a Figura 15 também apresenta o mesmo funcionamento para estrutura de diretórios, que tiveram a mesma alteração.

Figura 14 - Estrutura de *tags* em arquivo



Fonte: Elaborado pelo autor (2023)

Figura 15 - Estrutura de *tags* em diretório



Fonte: Elaborado pelo autor (2023)

5 Resultados e Análise

Para poder demonstrar o funcionamento do sistema de arquivos e posteriormente de sua integração com o sistema de *tagging*, foi desenvolvido uma interface que através do uso de funções implementadas no sistema de arquivos é feita a simulação de um interpretador de comandos do Linux.

A interface fornece as funções que serão descritas e terão seu uso exemplificado a seguir. A partir da seção 5.1.2 (Comando: ls) nos exemplos apresentados, com exceção da formatação, o sistema foi previamente populado executando os comandos que estão disponíveis no arquivo commands.txt presente no repositório[3] do sistema, onde também estão as instruções para a compilação do programa no sistema Linux.

São mostrados exemplos do funcionamento referentes a cada funcionalidade após sua apresentação.

5.1 Resultados

3 Disponível em: https://gitlab.com/alvaro.alvinn/so-sistema-de-arquivos

Nesta seção serão apresentados os comandos que permitem a operação do sistema de arquivos, pois a partir deles é possível utilizar todas as funções que estão presentes no sistema de arquivos, assim como as funções que utilizam do sistema de *tagging*, a partir da seção 5.1.10 (Comando: ltags), assim mostrando os sistemas funcionando de forma integrada.

5.1.1 Formatação e Execução.

Para formatar um dispositivo é preciso executar o programa com três argumentos, sendo o primeiro `-f`, que indica que o programa irá iniciar em modo de formatação, o segundo é o tamanho da unidade de cada bloco que irá dividir o dispositivo, e o terceiro é um arquivo que representa o dispositivo que será formatado, conforme é comentado na seção 3.1.2 (Projeto e Implementação).

Uso: `./fs -f <tamanho de bloco> <nome do arquivo>`

```
./fs -f 512 disk.data
formatando...
disk.data formatado com sucesso!
```

Para executar o sistema, é preciso executá-lo passando como único argumento um dispositivo que tenha sido anteriormente formatado.

Uso: `./fs <nome do arquivo>`

```
./fs disk.data
Abrindo dispositivo...
Extraindo configuração do dispositivo...
SISTEMA RECUPERADO COM SUCESSO!
Para ajuda digite 'help', para sair digite 'exit'!
/$
```

5.1.2 Comando: `ls`

O comando `ls` lista todos os diretórios e arquivos no diretório atual. Os diretórios são exibidos em negrito para melhor distinção.

Uso: `ls`

```
/$ ls
Natureza
Alimentos
Cores
Profissões
Tecnologia
Documentos
/$
```

5.1.3 Comando: `touch`

O comando `touch` permite a criação de um novo arquivo de texto no diretório atual.

Uso: `touch <nome do novo arquivo>`

```
/$ touch novo_arquivo.txt
/$ ls
Natureza
Alimentos
Cores
Profissões
Tecnologia
Documentos
novo_arquivo.txt
/$
```

5.1.4 Comando: `echo`

O comando `echo` permite escrever em um arquivo de texto.

Uso: `echo <conteúdo a ser escrito entre aspas duplas> <nome do arquivo>`

```
/$ touch novo_arquivo.txt
/$ echo "conteudo do novo arquivo!" novo_arquivo.txt
/$
```

5.1.5 Comando: `cat`

O comando `cat` permite ler o conteúdo de um arquivo.

Uso: `cat <nome do arquivo>`

```
/$ cat novo_arquivo.txt
conteudo do novo arquivo!
/$
```

5.1.6 Comando: `mdkir`

O comando `mkdir` permite a criação de um diretório no diretório atual.

Uso: `mkdir <nome do novo diretório>`

```
mkdir novo_diretório
/$ ls
Natureza
Alimentos
Cores
Profissões
Tecnologia
Documentos
novo_arquivo.txt
novo_diretório
/$
```

5.1.7 Comando: `rm`

O comando `rm` permite deletar um arquivo ou diretório.

Uso: rm <caminho do arquivo ou diretório que sera removido>

```
/$ ls
Natureza
Alimentos
Cores
Profissões
Tecnologia
Documentos
novo_arquivo.txt
novo_diretório
/$ rm novo_arquivo.txt
/$ rm novo_diretório
/$ ls
Natureza
Alimentos
Cores
Profissões
Tecnologia
Documentos
/$
```

5.1.8 Comando: cd

O comando cd permite acessar um diretório.

Uso: cd <caminho do diretório desejado>

```
/$ ls
Natureza
Alimentos
Cores
Profissões
Tecnologia
Documentos
/$ cd Documentos
/Documentos/$ ls
Artigos
Slides
```

```
Docs
Apresentações
/Documentos/$ cd ..
/$ ls
Natureza
Alimentos
Cores
Profissões
Tecnologia
Documentos
/$ cd Documentos/Artigos
/Documentos/Artigos/$ ls
Arte
Arquitetura
Arte_e_Arquitetura.pdf
Matemática
Computação
Biologia
Geografia
/$
```

5.1.9 Comando: `namefind`

O comando `namefind` faz uma busca recursiva a partir do diretório atual exibindo todos os arquivos encontrados que possuem o termo pesquisado em seu nome.

Uso: `namefind <nome de arquivo para busca>`

```
/Documentos/Artigos/$ namefind arte.pdf
/Documentos/Artigos/Arte/arte.pdf
/Documentos/Artigos/$ cd ...
/$ namefind arte.pdf
/Documentos/Artigos/Arte/arte.pdf
/$
```

5.1.10 Comando: `ltags`

O comando `ltags` lista as *tags* que estão relacionadas com um arquivo, o nome do arquivo é passado como argumento.

Uso: `ltags <nome do arquivo>`

```
/$ cd Documentos/Artigos/Arte
/Documentos/Artigos/Arte/$ ls
arte.pdf
/Documentos/Artigos/Arte/$ ltags arte.pdf
arte
/Documentos/Artigos/Arte/$
```

5.1.11 Comando: `ctags`

O comando `ctags` remove todas as *tags* que estão associadas com um arquivo, o nome do arquivo é passado como argumento.

```
Uso: ctags <nome de arquivo>
```

```
/Documentos/Artigos/Arte/$ ctags arte.pdf
/Documentos/Artigos/Arte/$ ltags arte.pdf
/Documentos/Artigos/Arte/$
```

5.1.12 Comando: `addtag`

O comando `addtag` adiciona uma nova *tag* a um arquivo, recebe dois argumentos, primeiro a palavra-chave da *tag* e segundo o nome do arquivo com o qual a *tag* vai ser associada.

Uso: `addtag <palavra-chave> <nome do arquivo>`

```
/Documentos/Artigos/Arte/$ addtag arte arte.pdf
/Documentos/Artigos/Arte/$ addtag nova_tag arte.pdf
/Documentos/Artigos/Arte/$ ltags arte.pdf
```

```
arte
nova_tag
/Documentos/Artigos/Arte/$
```

5.1.13 Comando: `rmtag`

O comando `rmtag` remove uma *tag* de um arquivo, o comando recebe dois argumentos, primeiro a palavra-chave da *tag* e segundo o nome do arquivo do qual a *tag* sera desassociada.

Uso: `rmtag <palavra-chave> <nome do arquivo>`

```
/Documentos/Artigos/Arte/$ rmtag nova_tag arte.pdf
/Documentos/Artigos/Arte/$ ltags arte.pdf
arte
/Documentos/Artigos/Arte/$
```

5.1.14 Comando: `tagfind`

O comando `tagfind` faz uma listagem do caminho completo de todos os arquivos que estão associados com determinada *tag*, recebe como argumento a palavra-chave da *tag* desejada.

Uso: `tagfind <nome de tag para busca>`

```
/$ tagfind arte
/Documentos/Artigos/ArqArte.pdf
/Documentos/Artigos/Arte/arte.pdf
/$ tagfind arquitetura
/Documentos/Artigos/ArqArte.pdf
/Documentos/Artigos/Arquitetura/arquitetura.pdf
/$
```

5.1.15 Comando: `find`

O comando `find` executa o comando `tagfind` e `namefind` simultaneamente, dessa forma fazendo a listagem dos arquivos encontrados, entretanto, dessa forma os arquivos encontrados com base nas *tags* são identificados com "(TAG)" escritos ao seu lado.

Uso: `find <tag ou nome de arquivo para busca>`

```
/$ find arte
/Documentos/Artigos/ArqArte.pdf (TAG)
/Documentos/Artigos/Arte/arte.pdf (TAG)
/Documentos/Artigos/Arte/arte.pdf
/$ find arquitetura
/Documentos/Artigos/ArqArte.pdf (TAG)
/Documentos/Artigos/Arquitetura/arquitetura.pdf
(TAG)
/Documentos/Artigos/Arquitetura/arquitetura.pdf
/$
```

5.2 Análise

Na seção 5.2.1 (Comparação com Trabalhos Correlatos) será feita uma análise comparando o trabalho realizado com os trabalhos correlatos apresentados na seção 3.2.1 (Revisão Bibliográfica), e na seção 5.2.2 (Desempenho) será feita uma análise do desempenho e da aplicabilidade do sistema em uma escala maior.

5.2.1 Comparação com Trabalhos Correlatos

O sistema final permite com que o sistema de arquivos seja utilizado de forma independente, e até que seja iniciado, com a primeira *tag* sendo adicionada, o sistema de *tagging* não ocupa mais espaço no sistema, já que os blocos que guardam suas informações só são alocados à medida que *tags* são adicionadas.

O mesmo ocorre para cada arquivo, o espaço armazenado por ele quando não possui uma *tag* é o mesmo de quando o sistema não era integrado ao sistema de tag, com a adição do

sistema sendo necessário apenas uma variável que indica o bloco de *tags*, informação que poderia estar presente em outra estrutura, que depende da implementação.

Fazendo uma comparação com os trabalhos correlatos apresentados na seção 3.2.1 (Revisão Bibliográfica) temos a vantagens apresentadas a seguir:

TagFS (BLOEHDORN et al., 2006)**:** O TagFS é implementado sobre um webDAV, ou seja, pode ser usado somente através da abertura de um servidor de arquivos para acesso, além disso, não é possível adicionar/remover *tags* explicitamente, já que as *tags* são extraídas automaticamente de acordo com os diretórios do arquivo movido para o sistema. No sistema apresentado, além de teoricamente ser possível utilizar em um servidor de arquivos, visto que basta que o servidor esteja utilizando o sistema, também é possível definir explicitamente as *tags* associadas com cada arquivo.

hFAD (SELTZER; MURPHY, 2009)**:** O hFAD é somente uma proposta inicial de arquitetura, e não é apresentada nenhuma implementação, sua arquitetura busca extinguir totalmente a parte fundamentalmente hierárquica do sistema de arquivos, utilizando no lugar um sistema totalmente novo de armazenamento de objetos. O sistema apresentado busca integrar a funcionalidade de *tags* a um sistema hierárquico já existente e não basear todo um novo sistema em *tags*.

Semantic File System (GIFFORD et al., 1991)**:** O sistema apresentado, *Semantic File System*, apresenta uma interface sobreposta ao sistema de arquivos com conjuntos de dados que representam metadados adicionais por meio de campos de chave-valor, o sistema apresentado se baseia de analisadores customizados para a extração de dados de cada tipo de arquivo enquanto o sistema apresentado nesse trabalho apresenta uma estrutura mais genérica de *tags* que permite a adaptação para qualquer tipo de arquivo, além disso, por ser uma estrutura contida junto ao sistema de arquivos base, é possível que qualquer outro aplicativo externo possa usar suas funcionalidades.

O Quadro 1 apresenta uma comparação entre os principais pontos dos trabalhos correlatos e o sistema que é apresentado neste trabalho. O método indica o que é o cada um dos trabalhos. A abordagem semântica indica qual o artefato semântico utilizado pelo trabalho. A estrutura semântica indica como a abordagem semântica é utilizada. Com implementação indica se o sistema possui ou não uma implementação. E na tabela, a sigla SA indica sistema de arquivos.

Quadro 1 - Quadro comparativo entre trabalhos

	TagFS	SFS	hFAD	Este trabalho
Método	Servidor de arquivos	Interface sobre o SA	Novo SA	Integração com o SA
Abordagem semântica	Tags	Campos chave-valor	OSD (*Object-based Storage Device*)	Tags
Estrutura semântica	Tags implícitas (diretórios)	Programas customizados para preencher campos	Banco de dados	Tags explícitas
Com Implementação	Sim	Sim	Não	Sim

Fonte: Elaborado pelo autor (2023)

5.2.2 Desempenho

O objetivo do sistema desenvolvido é mostrar a possibilidade da implementação de um sistema de *tagging* junto a um sistema de arquivos, sendo considerado um protótipo. Ao analisar a estrutura implementada é possível identificar gargalos à medida que mais *tags* são adicionadas aos arquivos, principalmente relacionado ao uso de memória e ao acesso ao disco.

O sistema, em seu estado atual, armazena toda a tabela de alocação, árvore de *tags* e listas de arquivos associados às *tags* em memória. A ideia de armazenar toda tabela de alocação em memória já é ultrapassada, por exemplo, ao utilizar o sistema em um HD de 500 GB dividido em blocos de 512 *bytes* a tabela de alocação ocuparia quase 4 GB o que é uma quantidade inviável.

A estrutura de árvore de *tag* é transformada em *buffer* e escrita nos blocos do dispositivo a cada modificação na árvore, visto que por ser uma estrutura auto-organizada, uma inserção ou remoção pode modificar várias partes da estrutura da árvore, não sendo possível fazer uma atualização parcial da estrutura no disco, ou seja, esse processo ocorre a cada inserção de nova *tag* ou remoção de *tag* existente.

Em um sistema com poucas *tags* isso não é um problema, mas a medida que o sistema cresce o impacto se torna maior. Considerando um sistema que possua 10000 *tags*, e que o tamanho do nome da *tag* seja de 32 caracteres, isso representa 320000 *bytes*, logo, em us sistema com tamanho de bloco de 512 *bytes* é necessário escrever 625 blocos em disco para cada operação, tendo em vista que o acesso ao disco é uma operação demorada em relação ao tempo de acesso à memória, o ideal é que seja realizada o mínimo de vezes necessária, ainda mais que operações em excesso diminuem a vida útil da unidade.

6 Conclusões e Trabalhos Futuros

A implementação de um sistema de arquivos com um sistema de *tagging* integrado permite com que possa haver mais caminhos para chegar até cada arquivo, por meio de novas relações semânticas através do uso de *tags*.

Por meio da integração de um sistema ao outro foi possível manter as funcionalidades padrões do sistema intactas, somente adicionando novas funções através do sistema de tags, que exigiu poucas modificações ao sistema base e funciona em uma estrutura separada a do sistema de endereçamento padrão dos arquivos através do sistema de diretórios.

A implementação do sistema apresentado neste trabalho é somente um protótipo, tendo em vista que um sistema que simula as funcionalidades de um sistema de arquivos, já que executa as operações se leitura e escrita sobre o sistema de arquivos do sistema no qual está sendo executado. Portanto, existem diversas possíveis otimizações que podem ser feitas ao sistema, assim como adequá-lo a um modelo de endereçamento mais moderno como o *I-node* que permite com que a tabela de alocação seja descentralizada e as suas informações sejam mantidas em memória somente conforme o necessário.

Existem também possibilidades para a otimização do uso de memória e acesso ao disco da estrutura de *tags*, especialmente com a árvore de *tags*, que é totalmente gravada em disco a cada modificação. Uma alternativa de implementação para reduzir o uso de memória e acesso ao disco seria o armazenamento a integrações com um sistema de *journaling*, que já está presente em sistemas modernos, dessa forma registrando as operações e realizando ocasionalmente a escrita no disco. Uma maneira de diminuir o uso de memória do sistema de *tags* durante a execução do sistema seria a aplicação de um sistema semelhante ao *I-node* para o armazenamento da árvore, com partes da árvore armazenadas em blocos e realizada a leitura desses blocos a medida que essas partes precisam ser acessadas para a busca.

Além dessas melhorias, o sistema também poderia ser portado para o kernel, para ser realmente executado como um sistema de arquivos.

Funcionalidades adicionais podem ser adicionadas ao sistema de *tagging*. Uma delas é a recuperação referente a falhas, como as informações com relação as *tags* estão presentes tanto na árvore de *tag* quando em cada arquivo que possui uma *tag* associada, se houver

alguma corrupção em um desses locais, é possível utilizar o outro para restaurar as informações perdidas e manter a integridade.

Outra funcionalidade possível em trabalhos futuros é o desenvolvimento de um sistema que utilize o sistema de *tags* e obtenha informações referentes a cada arquivo para criar relações de *tags* de forma automática.

Referências

BLOEHDORN, Stephan et al. **Tagfs-tag semantics for hierarchical file systems**. Proceedings of the 6th International Conference on Knowledge Management (I-KNOW 06), Graz, Austria. p. 6-8, 2006.

GIFFORD, D. K. et al. **Semantic file systems**. ACM SIGOPS operating systems review, v. 25, n. 5, p. 16-25, 1 set. 1991.

MOGULL, J. C. **Representing information about files**. International Conference on Distributed Computing Systems, p. 432–439, 1 jun. 1986.

PIGEON, Steven. **Compact Tree Storage**, Harder, Better, Faster, Stronger. [*S.l.*], 7 abr. 2009. Disponível em: https://hbfs.wordpress.com/2009/04/07/compact-tree-storage/. Acesso em: 26 ago. 2023.

SELTZER, M.; MURPHY, N. A. **Hierarchical file systems are dead**. p. 1–1, 18 maio 2009.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operating system concepts**. Hoboken, N.J: Wiley, 2018.

STANDISH, T. A. **Data Structure Techniques**. [*S.l.*] Addison Wesley Publishing Company, 1980.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. 4. ed. Harlow: Pearson Education, 2015.