João Vitor Moraski Lunkes

Projeto
Implementação de Sistema de Arquivos EXT2

Relatório técnico de atividade prática solicitado pelo professor Rodrigo Campiolo na disciplina de Sistemas Operacionais do Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Universidade Tecnológica Federal do Paraná – UTFPR

Departamento Acadêmico de Computação – DACOM

Bacharelado em Ciência da Computação – BCC

Campo Mourão

Março / 2025

Sumário

Lista de ilustrações **2**

Lista de tabelas **2**

1 Introdução 4
2 Descrição da atividade 4
2.1 Operações a serem implementadas: 4
2.2 Limitações e Possíveis Pontos de Falha 5
3 Métodos 5
3.1 Estrutura de dados e Gerenciamento de estado 5
3.1.1 Struct base 6
3.2 Funções auxiliares 6
3.2.1 Navegação e resolução de caminhos 6
3.2.2 Algoritmos e Alocação e Desalocação 7
3.2.3 Gerenciamento de entradas de diretório 7
3.3 Makefile e tratamento das entradas do terminal 8
3.3.1 Main 8
4 Resultados e Discussões 9
4.1 Exemplo de funcionamento 9
5 Bugs Conhecidos 12
6 Divisão 12
7 Conclusão 13
8 Referências 14

Lista de ilustrações

Figura 1 – Execução do comando info ao iniciar a imagem. 9
Figura 2 – Execução do comando info após realizar o comando mkdir. 9
Figura 3 – Execução do comando ls após a criação de novos diretórios. 10
Figura 4 – Execução do comando info após realizar o comando rmdir. 10
Figura 5 – Execução da verificação de corrupção da imagem. 11

Lista de tabelas

Tabela 1 – Tabela de divisão de trabalho 12

1 Introdução

O sistema de arquivos EXT2(Second Extended File System), criado por Rémy Card possui uma implementação simples e facil, uma eficiente gerencia de espaço no disco, foi criado como uma melhoria do Ext, inspirado pelo BSD e o primeiro a ser utilizado de forma comercial para Linux. O objetivo desse trabalho foi implementar um *shell* para interações com o sistema Ext2 que esta sendo usado na imagem *myext2image.img*.

2 Descrição da atividade

Implementar as estruturas de dados e operações para manipular uma imagem (.iso/.img) de um sistema de arquivos EXT2. As operações devem executar a partir de um terminal próprio(shell).

2.1 Operações a serem implementadas:

- info: exibe informações do disco e do sistema de arquivos
- cat <file>: exibe o conteúdo de um arquivo no formato texto.
- attr <file | dir>: exibe os atributos de um arquivo (file) ou diretório (dir).
- cd <path>: altera o diretório corrente para o definido como path.
- ls: lista os arquivos e diretórios do diretório corrente.
- pwd: exibe o diretório corrente (caminho absoluto).
- touch <file>: cria o arquivo file com conteúdo vazio.
- mkdir <dir>: cria o diretório dir vazio.
- rm <file>: remove o arquivo file do sistema.
- rmdir <dir>: remove o diretório dir, se estiver vazio.
- rename <file> <newfilename> : renomeia arquivo file para newfilename.
- cp <source_path> <target_path>: copia um arquivo de origem para destino.
- mv <source_path> <target_path>: move um arquivo de origem para destino.

2.2 Limitações e Possíveis Pontos de Falha

Como restrição foi dito que não seria permitido usar funções do sistema para executar os comandos na imagem, como system(), exec() e não seria possível utilizar as estruturas já prontas do EXT2.

Como simplificação foi dito que não seria necessário lidar com multiplos diretorios nos comandos, ou o caminho inteiro ate um arquivo. Considerar que não havia arquivos maiores que 64MiB e o tamanho do bloco como 1024, e diretorios tendo como tamanho de apenas 1 bloco.

3 Métodos

3.1 Estrutura de dados e Gerenciamento de estado

As estruturas de dados definidas no arquivo `ext2-implext2_structs.h`, foram criadas com base na documentação do EXT2(POIRIER, 2025) e na biblioteca de referência(MEEKS, 2025), no arquivo `ext2_fs.h`. Um ponto crucial na implementação foi a utilização da diretiva `__attribute__((packed))`(C, 2025) na definição das structs.

```
typedef struct __attribute__((packed)) {
    uint32_t inode;
    uint16_t rec_len;
    uint8_t name_len;
    uint8_t file_type;
    char  name[255];
} dir_entry;
```

Por padrão, o compilador da linguagem C lê a struct inserindo alguns bytes de preenchimento(padding) entre os campos da struct, para garantir o alinhamento dos valores na memoria e melhorar o acesso pela CPU. Porém, no EXT2 as estruturas do sistema de arquivos é feito de forma continua, sem presencia de bytes de preenchimento.

A diretiva `__attribute__((packed))` instrui o compilador a desabilitar esta otimização com o alinhamento, forçando o compilador a alocar os campos da `struct` um imediatamente após ao outro, sem bytes de preenchimento, garantindo que a representação da `struct` em memória seja um espelho exato, byte por byte da estrutura no disco.

3.1.1 Struct base

Para a facilidade da implementação das manipulações, e evitar que alguns dados sensíveis fossem acessados toda hora, foi criado uma `struct` base chamada de `ext2_info`, que possui alguns campos principais para o controle dos dados.

```
typedef struct {
    super_block sb; // variavel do super bloco
    group_desc* group_desc_array; // array de descritores de grupo
    unsigned int block_size; // tamanho do bloco em bytes (ja calculado de s_log_block_size)
    unsigned int num_block_groups; // numero total de grupos de blocos (calculado do sb)
    unsigned int current_dir_inode; // Inode do diretorio corrente (para cd, ls, pwd)
    char current_path[256]; // path atual
    int fd; // descritor de arquivo da imagem(aberto no inicio e fechado ao final)
} ext2_info;
```

3.2 Funções auxiliares

Uma das otimizações feitas nesse trabalho foi a criação de funções auxiliares para evitar a duplicação de codigo de busca de `inodes, blocos, path`. Estas funções podem ser encontradas no arquivo `ext2-impl/ext2-fs-methods.c`.

A função `load_superblock` foi criada para que seja feito o carregamento do superblock na memoria, começa iniciando a `struct` criado para representar o superbloco e indo ate a posição da memoria de seus dados, que seria o primeiro offset de 1024. Após a leitura e verificação do campo `s_magic` (indica se é ou não um EXT2) é feito o preenchimento dos campos `ext2_info->sb e block_size`.

A função `load_group_desc` foi criada para que seja feito o carregamento dos descritores de grupo na memoria, começa calculando o número de grupos de blocos presentes na imagem, após isso é criado com o `malloc` o array para armazenar os descritores, após a leitura destes descritores, que ficam logo após o superbloco(2048) é definido o caminho inicial () e o número do `inode` do atual diretório, sendo indicado como 2 porque é o diretório raiz.

3.2.1 Navegação e resolução de caminhos

A função `read_inode_by_number` foi criada para facilitar a leitura de `inodes` do sistema de arquivos, é passado o número do `inode` para a função e baseado neste número é calculado qual o grupo deste `inode`, e então é lido as informações do `inode` no disco retornada a `struct` preenchida.

A função `find_inode_by_path` foi criada para facilitar a leitura de `inodes` do sistema de arquivos, é passado o caminho para o arquivo ou diretório e a função busca o `inode` relacionado a este caminho. Considera caso o caminho comece na raiz() ou a partir do diretório atual.

Similar a outros métodos, após ler o bloco de dados, começa a iterar baseado no tamanho das entradas de diretório procurando pelo caminho até chegar ao destino e informar o número do `inode` desejado.

A função `find_parent_inode_and_final_name` foi criada para facilitar a busca pelo `inode` pai e separar o nome do arquivo informado no caminho. Nesta função é feito algumas operações de verificação para que seja retirado o ultimo valor informado no caminho, por exemplo, em `/classicos/livros/dracula.txt` é retirado o `dracula.txt` e passado para o ponteiro de nome de arquivo passado para a função, e em seguida executa a função `find_inode_by_path`, passando o caminho que restou e por fim retornando o número do `inode` pai e preenchendo o nome do arquivo.

3.2.2 Algoritmos e Alocação e Desalocação

A função `write_inode_by_number` foi criada para facilitar a escrita de `inodes` do sistema de arquivos, é passado o número do `inode` para a função e baseado neste número é calculado qual o grupo deste `inode`, e então é persistido as informações do `inode` no disco usando a `struct` enviada para a função.

As funções `allocate_item e deallocate_item` foram criadas para generalizar o processo de criação e remoção de itens(blocos ou `inodes`), é passado o tipo(i ou b) e o número do item(para ser removido), então é feito a busca entre os descritores de grupos procurando por algum espaço livre ou pela posição do item informado.

Após achar o item, ou a posição vaga, é modificado no `bitmap de inode ou block` para indicar aquele espaço como utilizado(allocate) ou livre(deallocate). As operações de acrescentar ou diminuir a contagem de `inodes ou blocks` é feita em ambas a funções.

3.2.3 Gerenciamento de entradas de diretório

A função `add_dir_entry` foi criada para facilitar a criação de uma nova `dir_entry`, usada tanto por arquivos quanto para diretórios, é informado número do `inode` do pai(diretório antes) e do novo `inode` a ser adicionado, junto com o nome do arquivo, um indicador de tipo e uma variavel para controle do `dry_run`.

Faz a leitura da imagem e do bloco de dados do `inode` do pai, após isso é calculado o tamanho da nova entrada e procura por espaços aonde possa encaixar, se não encontrar,

coloca ao fim. O loop pelas entradas de diretório é feito usando a variavel indicadora de tamanho `rec_len`.

O `dry_run` é para verificar previamente se existe espaço suficiente naquele diretório para uma nova entrada.

A função `remove_dir_entry` foi criada para facilitar a remoção de uma `dir_entry`, usada tanto por arquivos quanto para diretórios, é informado número do `inode` do pai(diretório antes), junto com o nome do arquivo ao ser removido.

Faz a leitura da imagem e do bloco de dados do `inode` do pai, após isso é percorrido o bloco de dados procurando pela entrada que possua o mesmo nome, e ao encontrar é removida e salva no disco.

3.3 Makefile e tratamento das entradas do terminal

O projeto pode ser executado utilizando o comando:

```
1    make run
```

Para a recriação da imagem pode ser executado:

```
1    make image
```

E verificações se a imagem foi corrompida ou algum dado incorreto:

```
1    make verify-ext2
```

Algumas definições no `Makefile` foram feitas para facilitar o desenvolvimento, como adicionar as flags `-g -Wall` para avisos e depuração.

3.3.1 Main

No arquivo `main.c` foi implementando uma função para gerar a string do prompt, em homenagem ao tema do *ZSH*, `bira-theme`.

Foi implementado a leitura dos comandos utilizando da biblioteca `<readline/readline.h> e <readline/history.h>`, a biblioteca `readline` foi utilizada para facilitar algumas interações com o terminal, como controle pelas setas do teclado, atalhos para limpar a tela(comando clear) e o `history` foi adicionado junto com a `readline` para que salve o historico dos comandos em um arquivo, que é lido sempre que iniciado.

Inicialmente o terminal não conseguia lidar com entradas usando aspas, com a criação da função `parse_input` no arquivo `utils/utils.h/.c` ele foi capaz de lidar com entradas que usam de aspas, esta necessidade foi notada ao tentar executar o comando `cat` em um dos livros no diretório `/livros/classicos`, que possuía espaços entre seu nome. Esta função lida com as aspas indicando o inicio e fim do argumento com as diretivas de fim de *string* (\0), e salvando os ponteiros de inicio para a variável de argumentos.

4 Resultados e Discussões

Para a verificação do resultados foi utilizado o comando fornecido pelo professor `e2fsck -nvf`, este comando com as opções passadas mostram se a imagem tem algum problema de alocação, indicação de uso sem ter `inode` cadastrado e outros possíveis erros.

4.1 Exemplo de funcionamento

Inicialmente foi validado se os dados da imagem batiam com a do exemplo do professor.

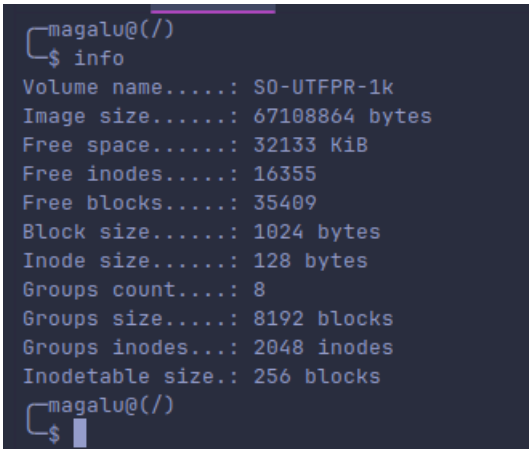

Figura 1 – Execução do comando info ao iniciar a imagem.

Para validar a operação do `mkdir`, múltiplos diretórios foram criados sequencialmente, como mostrado na imagem abaixo, foi executado o comando info para verificar os números de posições vagas e espaço livre, como observado na imagem, foi alterado os contadores de `Free Space e Free inodes`.

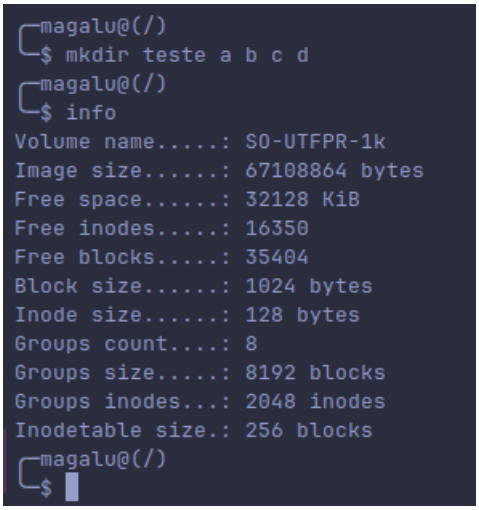

Figura 2 – Execução do comando info após realizar o comando mkdir.

Com isso, foi executado o comando `ls` para validar a criação dos diretórios e listagem com validação de tipos e tamanho.

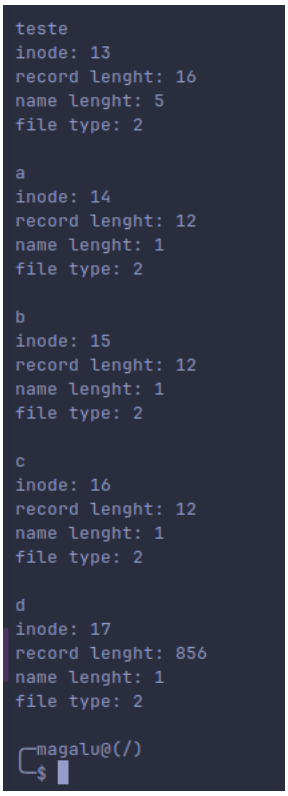

Figura 3 – Execução do comando ls após a criação de novos diretórios.

Em seguida, foi testado a remoção dos diretórios criados e logo em seguida a validação dos contadores novamente.

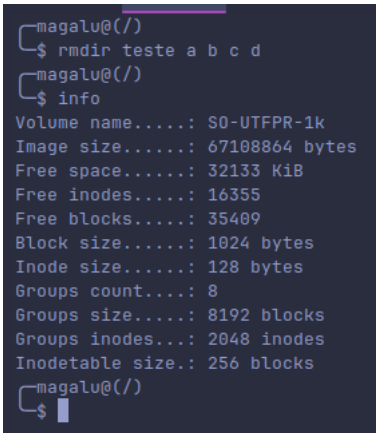

Figura 4 – Execução do comando info após realizar o comando rmdir.

Ao fim, o comando `e2fsck` foi executado com as opções `-nvf` para validação da integridade da imagem, contando os inodes, pastas, blocks, validação de bitmaps e valores de contagem.

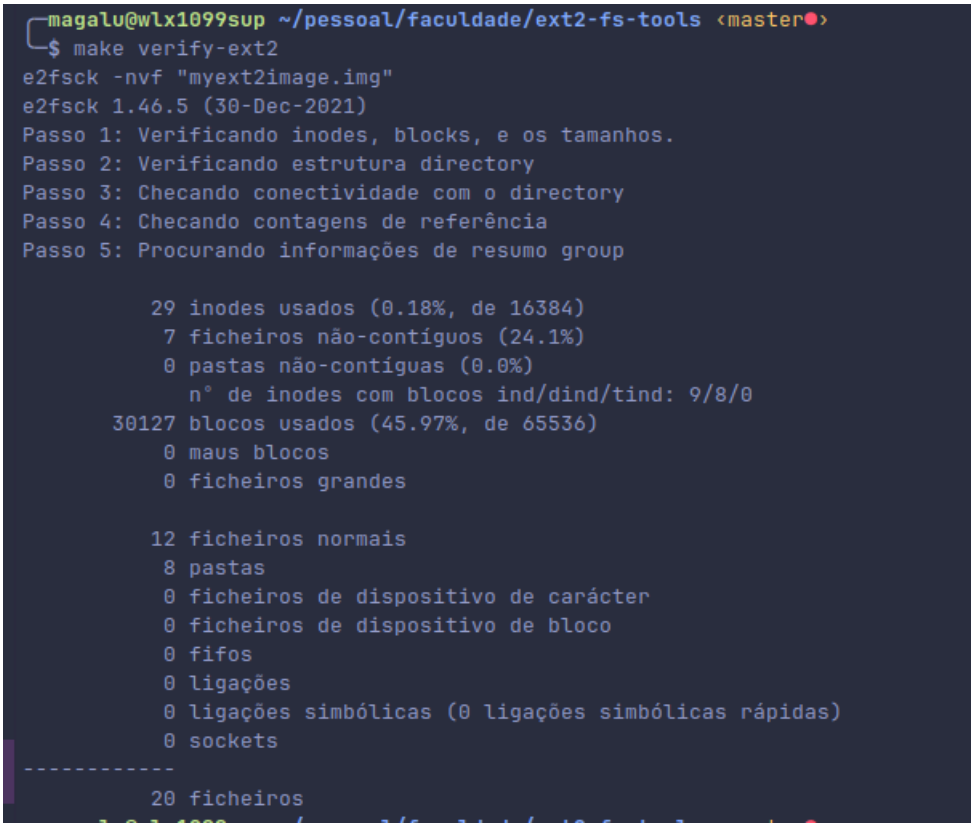

Figura 5 – Execução da verificação de corrupção da imagem.

A implementação das operações de manipulação do sistema de arquivos EXT2 apresentou diversos desafios que foram além da tradução da especificação para código, mas também uma analise sobre as funções e futuro do código para criar funções genéricas para que fossem usadas em diversos comandos. Como as alterações de adição ou remoção de novos arquivos/diretórios é algo que ocorre de forma parecida, foi pensado alguns métodos que já facilitariam esse trabalho para comandos futuros.

Um dos maiores desafios foi manter a integridade da imagem após o comando `mkdir` e `rmdir`. Usando a ferramenta `e2fsck` para indicar erros na imagem. Inicialmente os erros como "Unattached inode" e inconsistências nos bitmaps comuns. Durante a investigação destes erros foi possível definir que a causa do erro era a ordem que era feito a remoção de inodes ou blocos. Para a correção a operação de remoção do `dir_entry` foi feita primeiro, em seguida, a atualização dos metadados (contadores) da imagem, e por fim, os recursos físicos da imagem (inode e blocos) são desalocados, evitando assim, inodes "órfãos".

Outro ponto de investigação foi a contagem de links para os diretórios, foi um aprendizado importante para entender que a criação de um subdiretório exige a incre-

mentação de links para o diretório pai e que a remoção implica em uma redução deste valor, a ocorrencia deste erro no `e2fsck` também serviu para entender que ao remover um alvo ele também precisa ser decrementado.

Do ponto de vista da engenharia de software, o projeto exigiu decisões para gerenciar a complexidade e evitar a duplicação de código, utilizando de funções genéricas para a alocação e desalocação de itens, podendo ser inodes ou blocos.

5 Bugs Conhecidos

Durantes os testes finais feitos para validar o comportamento do terminal implementado baseado nas operações fornecidas pelo professor, não houve nenhum bug identificado. Porém a lógica das funções e algumas simplificações permitidas podem gerar algumas inconsistências na imagem.

Um dos problemas encontrados após a leitura final do código é que em alguns comandos, onde envolve a remoção da entrada de diretório, para depois remover inodes, blocos e outros dados, é possível que a entrada de diretório retorne algum erro e não faça rollback das ações feitas anteriormente, como decrementar o numero de links para os diretórios, ou arquivos removidos. Este erro foi contornado validando a existência do diretório/arquivo antes mesmo de fazer as operações de diminuir a contagem de links, porém, em uma eventual falha de alteração da imagem, é possível que ocorra algum erro de inconsistência, mas seria algo muito difícil de ocorrer, já que não foi indicado que deveria lidar com concorrência, que seria um dos casos possíveis de problema.

Algumas mensagens do tratamento de erro podem não seguir o padrão do Linux(Feito e testado em um Ubuntu), as vezes a ordem ou o conteúdo da mensagem.

6 Divisão

Pesquisa	Código	Documentação	Relatório
João Moraski	João Moraski	João Moraski	João Moraski

Tabela 1 – Tabela de divisão de trabalho

7 Conclusão

O presente trabalho teve o objetivo de implementar um shell para operações em cima de uma imagem utilizando o sistema de arquivos EXT2. Foram implementados os comandos propostos, sendo eles, comandos essenciais para a manipulação da imagem, como por exemplo: `cd, mkdir, rmdir, ls, touch, mv, cp`. Ao fim, o objetivo proposto foi alcançado com sucesso, criando um shell que pode interagir com a imagem no disco de forma consistente. A integridade da imagem foi verificada usando o comando passado pelo professor `e2fsck`, confirmando a confiabilidade das logicas dos algoritmos de alocação e desalocação.

Durante o desenvolvimento foi possível aprender diversas coisas de como os sistemas de arquivos realmente funcionam, entendendo os conceitos e como algumas operações são implementadas, também foi necessário um período para refletir a respeito das funções antes de implementar, usando uma estrategia parecida com TDD, primeiro foi pensado em como seria feito as ações para depois começar com as manipulações mais complicadas. Adicionalmente, o projeto exigiu um entendimento maior de conceitos de baixo nível da linguagem C, como manipulação de bits e bitmaps, gerencia de ponteiros e ate mesmo indicações para o compilador lidar de outra forma com uma struct.

Embora o objetivo do trabalho principal tenha sido concluído, existem diversas melhorias possíveis, sendo uma delas, a possibilidade de escrever em arquivos, sendo por meio de algum operador de escrita como nano ou vim, ou utilizando de envio de dados como parâmetros para comandos. Outra melhoria possível seria a implementação tanto do comando `mv` quanto `cp` para validar e executar caso seja uma operação interna da imagem, e não para fora. Em suma, o projeto proporcionou uma visão pratica de como é a complexidade e design de sistemas de arquivos, consolidando alguns conceitos teóricos em uma aplicação funcional e robusta, e também, proporcionou uma grande melhora no pensamento critico ao desenvolver funcionalidades para softwares reais.

8 Referências

ALTIERI, P. N. H. E. *The Ext2 Filesystem*. [S.l.], 2025. Disponível em: <https://www.science.smith.edu/~nhowe/262/oldlabs/ext2-.html>https://www.science.smith.edu/ nhowe/262/oldlabs/ext2.html. Acesso em: 5.6.2025. Nenhuma citação no texto.

C, G. *Packed Structures*. [S.l.], 2025. Disponível em: <https://www-.gnu.org/software/c-intro-and-ref/manual/html_node/Packed-Structures-.html>https://www.gnu.org/software/c-intro-and-ref/manual/html_node/Packed-Structures.html. Acesso em: 5.6.2025. Citado na página 5.

CARD STEPHEN TWEEDIE, T. T. R. *Design and Implementation of Second Extended Filesystem*. [S.l.], 2025. Disponível em: <https://e2fsprogs.sourceforge.net/ext2intro-.html>https://e2fsprogs.sourceforge.net/ext2intro.html. Acesso em: 5.6.2025. Nenhuma citação no texto.

MEEKS, M. *EXT2 Tools*. [S.l.], 2025. Disponível em: <https://github.com/mmeeks-/ext2tools>https://github.com/mmeeks/ext2tools. Acesso em: 5.6.2025. Citado na página 5.

POIRIER, D. *The Second Extended File System*. [S.l.], 2025. Disponível em: <https:-//www.nongnu.org/ext2-doc/ext2.html>https://www.nongnu.org/ext2-doc/ext2.html. Acesso em: 5.6.2025. Citado na página 5.

PROGRAMIZ. *C Pass Addresses and Pointers*. [S.l.], 2025. Disponível em: <https://www.programiz.com/c-programming/c-pointer-functions>https://www.programiz.com/c-programming/c-pointer-functions. Acesso em: 5.6.2025. Nenhuma citação no texto.

W3SCHOOLS. *C string strtok() function*. [S.l.], 2025. Disponível em: <https://www.w3schools.com/c/ref_string_strtok-.php>https://www.w3schools.com/c/ref_string_strtok.php. Acesso em: 5.6.2025. Nenhuma citação no texto.

WIKIPEDIA. *Ext2 Wikipedia*. [S.l.], 2025. Disponível em: <https://pt.wikipedia.org-/wiki/Ext2>https://pt.wikipedia.org/wiki/Ext2. Acesso em: 5.6.2025. Nenhuma citação no texto.