Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação de um Sistema de Arquivos simplificado

William - 5384

Projeto Final referente à disciplina Fundamentos de Sistemas Operacionais

UnB 2025

Resumo: Este trabalho, desenvolvido como projeto final da disciplina de Fundamentos de Sistemas Operacionais, na Universidade de Brasília, propõe a simulação de um Sistema de Arquivos com o objetivo de criação de diretórios e criar, mover e apagar arquivos.

Palavras-chave: Sistemas Operacionais; Sistemas de Arquivos.

1 Introdução

Um sistema de arquivos é uma estrutura fundamental em computação, responsável por organizar, armazenar e gerenciar dados em dispositivos de armazenamento, como discos rígidos ou memórias flash. Ele permite a criação, exclusão, leitura e escrita de arquivos e diretórios, além de fornecer mecanismos para navegar e manipular a hierarquia de dados. Em sistemas operacionais modernos, um sistema de arquivos é uma camada essencial que garante a persistência e o acesso eficiente aos dados.

Neste contexto, a linguagem C oferece uma base poderosa para implementar sistemas de arquivos simples, permitindo entender os conceitos fundamentais por trás dessa tecnologia. Embora a linguagem C não possua bibliotecas nativas para manipulação de sistemas de arquivos complexos (como os sistemas de arquivos de um sistema operacional), é possível simular um sistema de arquivos básico usando estruturas de dados.

Os conceitos fundamentais que nortearam o desenvolvimento e a implementação deste projeto foram:

1.1 Estruturas de Dados

- **Structs**: A estrutura FSItem é usada para representar arquivos e diretórios, contendo campos como nome, tipo, ponteiro para o pai, lista de filhos.
- **Árvore Hierárquica**: O sistema de arquivos é modelado como uma árvore, onde cada diretório pode conter outros diretórios ou arquivos (nós filhos).

1.2 Alocação Dinâmica de Memória

- **malloc e free:** Usados para alocar e liberar memória para novos arquivos/diretórios.
- **Gerenciamento de Memória:** É necessário garantir que toda memória alocada seja liberada corretamente para evitar vazamentos.

1.3 Manipulação de Strings

- **strncpy:** Usado para copiar nomes de arquivos/diretórios de forma segura, evitando estouro de buffer.
- **strcmp:** Compara strings para verificar se um arquivo/diretório já existe.

1.4 Recursão e Iteração

- **Iteração sobre Listas:** Percorre a lista de filhos de um diretório para operações como ls, rm, cp.
- **Navegação Hierárquica:** O comando cd permite navegar entre diretórios, alterando o ponteiro current_dir.

1.5 Modularização e Funções

- **Funções Especializadas:** Cada funcionalidade (ex: mkdir, touch, rm) é implementada em uma função separada, seguindo o princípio de responsabilidade única.
- **Reutilização de Código:** Funções como create_item são reutilizadas para criar arquivos e diretórios.

1.6 Controle de Acesso

- **Permissões Simples**: O campo is_locked na estrutura FSItem simula permissões de acesso, bloqueando ou desbloqueando operações como exclusão.

1.7 Interface de Usuário

- **Feedback Visual**: Mensagens coloridas (✓ verde para sucesso, ✗ vermelho para erros) e ícones (📁 para pastas, 📄 para arquivos) melhoram a experiência do usuário.
- **Prompt Interativo**: O prompt exibe o diretório atual e aguarda comandos do usuário.

1.8 Histórico de Comandos

- **Armazenamento de Comandos**: O histórico de comandos é armazenado em um array de strings (command_history).
- **Exibição do Histórico**: O comando history lista todos os comandos já executados.

1.9 Simulação de Operações de Arquivos

- **Criação de Arquivos/Diretórios**: Simulada com a alocação de estruturas FSItem.
- **Exclusão de Itens**: Remove itens da lista de filhos e libera memória.
- **Cópia e Movimentação**: Simula operações de cópia e movimentação de arquivos/diretórios.

1.10 Tratamento de Erros

- **Verificação de Limites**: Verifica se o número máximo de arquivos/diretórios foi atingido.

- **Validação de Entrada**: Verifica se um arquivo/diretório já existe antes de criá-lo.
- **Feedback de Erro**: Mensagens de erro claras ajudam o usuário a entender o que deu errado.

1.11 Conceitos de Sistemas Operacionais

- **Sistema de Arquivos Hierárquico**: Inspirado em sistemas como Unix/Linux, com diretórios e subdiretórios.
- **Comandos Básicos**: Implementa comandos semelhantes aos de um terminal Unix (mkdir, cd, ls, rm, etc.).
- **Gerenciamento de Recursos**: Simula o gerenciamento de recursos como memória e espaço em disco.

2. Formalização do Problema

A ideia principal deste projeto foi formatar um sistema de arquivos virtual simples que envie mensagens ao usuário durante operações como criação, seleção e movimentação de arquivos e pastas.

Foi definido uma estrutura básica de sistema de arquivos usando uma estrutura de árvore onde cada nó representa um diretório ou arquivo. Cada diretório pode ter uma lista de filhos, que podem ser arquivos ou outros diretórios.

Algumas funções foram formatadas para criar diretórios, navegar entre eles, listar conteúdo, mover itens e executar deleções. Além disso, cada operação deve gerar uma mensagem para o usuário, como confirmações ou erros.

O projeto inicial era formatar um sistema virtual, onde o acesso ao disco será implementado na memória (usando estruturas de dados) integrando em disco real, num projeto futuro. Uma *struct* para representar um item do sistema de arquivos, contendo nome, tipo (arquivo ou diretório), ponteiro para o diretório pai e lista de filhos, se for um diretório, foi implementado inicialmente.

A manutenção do controle do diretório atual para que o usuário possa navegar e uma variável global apontando para o diretório atual foi criada para uma melhor performance do sistema de arquivos.

Mensagem, em cada função, após realizar a operação, foi implementado para indicar o sucesso ou falha em determinada ação executada. Por exemplo: ao criar um diretório, imprimir "Diretório 'nome' criado com sucesso." Se o diretório já existir, informar o erro.

O link do código encontra-se em https://github.com/wilxavier/fso no arquivo virtualfsImproved.c. Para executá-lo instale o gcc, em seguida digite gcc -o filesystem virtualfsImproved.c e execute no terminal o comando ./filesystem.

3. Descrição do Algoritmo

O algoritmo simula um sistema de arquivos virtual, utilizando uma estrutura hierárquica de diretórios e arquivos. Ele oferece operações básicas como criação, navegação, listagem, remoção, cópia e controle de permissões, com feedback interativo para o usuário. É uma implementação simples, mas funcional, que demonstra conceitos fundamentais de sistemas de arquivos formatados na linguagem C.

3.1 Análise dos dados

Os dados gerados durante a execução do código são principalmente instâncias da estrutura FSItem, que representam arquivos e diretórios, e um histórico de comandos executados. O código manipula esses dados para simular operações básicas de um sistema de arquivos, como criação, remoção, cópia, movimentação e listagem de itens. A Figura 1 mostra uma sequência específica de comandos.

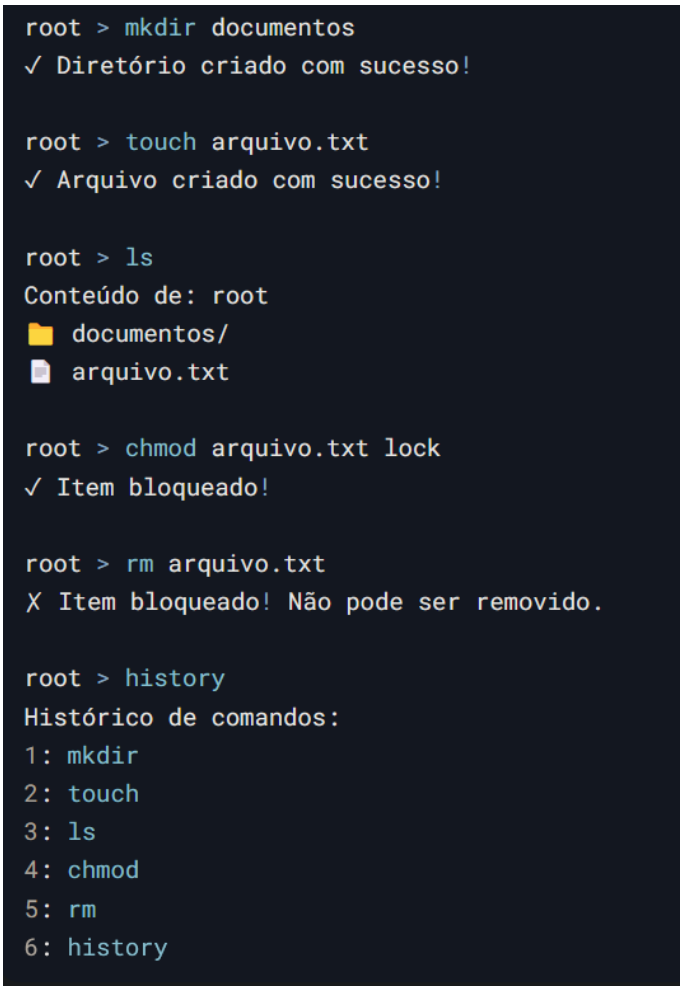

Figura 1 - Feedback para o usuário

4. Conclusão

Os resultados do projeto demonstram que o sistema implementado alcançou com êxito o objetivo estabelecido. Esta meta gira em torno da implementação de uma estrutura de sistema de arquivos e diretórios com o intuito de praticar alguns conceitos discutidos no curso de Fundamentos de Sistemas Operacionas. O código implementado executa instâncias da estrutura FSItem, onde arquivos e diretórios são representados, e todo histórico de comandos executados. O código faz a manipulação desses dados para simular operações básicas de um sistema de arquivos, como criação, remoção, cópia, movimentação e listagem de itens. Embora funcional, o sistema tem limitações que serão abordadas em versões futuras para melhorar sua utilidade e robustez.

5. Referências Bibliográficas

1. BEN-ARI, M. (2006). *Principles of Concurrent and Distributed Programming*, 2a ed. Addison-Wesley.

2. Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10a ed.). Wiley.