
UNIVERSIDADE DO VALE DO ITAJAÍ (UNIVALI)

DISCIPLINA: Sistemas Operacionais

PROFESSOR: Michael D C Alves

CURSO: Ciência da Computação

Trabalho M3

TRABALHO AVALIATIVO: Implementação e Análise de um Mini-Sistema de Arquivos em Memória em C

1. Introdução

Este trabalho tem como objetivo aprofundar o entendimento dos conceitos fundamentais de sistemas de arquivos em Sistemas Operacionais, extrapolando o uso de comandos e ferramentas existentes para a simulação de suas estruturas e operações internas. Os sistemas de arquivos são componentes críticos que gerenciam como os dados são armazenados, acessados e protegidos em um dispositivo de armazenamento secundário, como o disco. Eles fornecem uma interface de usuário para armazenamento e um mapeamento lógico para físico, permitindo que os dados sejam armazenados, localizados e recuperados facilmente.

Um arquivo é um tipo de dado abstrato, caracterizado por atributos como nome, identificador, tipo, localização, tamanho e proteção, além de informações de data e hora.

As operações básicas com arquivos incluem criar, escrever, ler, reposicionar, excluir e truncar. Além disso, o gerenciamento de permissões é essencial para controlar quem pode fazer o quê com um arquivo.

2. Objetivo Geral

Propor e implementar um sistema de gerenciamento de arquivos simplificado (em memória ou em um arquivo simulado de disco) em **linguagem C/Java/Phyton** que demonstre os conceitos fundamentais de arquivos, diretórios, “metadados” e controle de acesso, simulando a operação de um sistema de arquivos real em um ambiente “Linux”.

3. Objetivos Específicos

Para alcançar o objetivo geral, os seguintes objetivos específicos devem ser cumpridos:

- 3.1. Modelagem da Estrutura de Diretórios:**

- o** Implementar uma estrutura de dados (e.g., árvore binária, lista encadeada) que represente um **diretório estruturado por árvores**.

- o Permitir a criação de diretórios (mkdir) e a navegação entre eles (cd).
- **3.2. Representação e Gerenciamento de Arquivos e Metadados:**
 - o Criar uma estrutura para representar um **File Control Block (FCB)** para cada arquivo. Este FCB deve armazenar atributos como: nome (string), tamanho (inteiro), tipo (enum ou inteiro para dados: numérico, caractere, binário; programa), data de criação/modificação/acesso (time_t), um ID único (**simulando um inode**) (inteiro), e as **permissões de acesso** (inteiro ou bitmask).
 - o Implementar operações básicas como criar arquivo (touch), escrever conteúdo (echo > ou similar), ler conteúdo (cat ou similar), copiar (cp), mover/renomear (mv) e excluir (rm) - todas realizadas **internamente na estrutura de dados simulada**, não no sistema de arquivos real.
- **3.3. Controle de Acesso e Permissões:**
 - o Implementar um mecanismo de **proteção** baseado nas permissões RWX (Read, Write, Execute) para as três classes de usuários: proprietário (owner), grupo (group) e outros (public). Utilizar máscaras de bits ou operadores bit a bit para gerenciar as permissões (e.g., chmod numérico).
 - o O simulador deve permitir definir e exibir essas permissões.
 - o A cada tentativa de operação (ler, escrever, executar), o simulador deve verificar as permissões do "usuário atual" (que pode ser uma variável global simples) e permitir ou negar a operação com uma mensagem apropriada, simulando o conceito de bloqueio obrigatório.
- **3.4. Simulação de Alocação de Blocos**
 - o A simulação conceitual de **blocos de dados** (sugestão: um array de caracteres char[] para representar o "disco" simulado) e um método de **alocação de arquivos** (alocação contínua ou indexada simplificada).
 - o Demonstrar como o conteúdo de um arquivo é dividido em blocos e como esses blocos são "alocados" e referenciados pela estrutura do FCB/inode simulado.

4. Metodologia e Ferramentas

O trabalho pode ser desenvolvido em **linguagem C/Java/Phyton** e testado preferencialmente em um ambiente Linux ou equivalente para garantir a compreensão e a validação dos conceitos de sistemas de arquivos Unix-like.

- **4.1. Linguagem de Programação:**

- Poderão utilizar os conceitos de structs, ponteiros, alocação dinâmica.
Caso em C, memória (malloc, free), e manipulação de strings para construir as estruturas do sistema de arquivos simulado.

- **4.2. Ambiente de Desenvolvimento e Testes:**

- **Windows Subsystem for Linux (WSL):** É a opção mais indicada para usuários de Windows. Permite compilar e executar programas C em um ambiente Linux completo, o que é crucial para testar e comparar o simulador com o comportamento real dos comandos Linux (ls -l, stat, chmod, etc.).
- **Máquina Virtual (VM) com Linux:** (e.g., VirtualBox, VMware com Ubuntu, Fedora). Uma excelente opção para criar um ambiente Linux isolado, compilando e executando o código C diretamente no SO.
- **Terminal Linux:** Essencial para compilar o código (com gcc ou clang), executar o programa e interagir com a linha de comando do simulador.

- **4.3. Ferramentas de Compilação e Edição:**

- **Compilador C:** GCC (GNU Compiler Collection) ou Clang são os compiladores padrão para C em ambientes Linux.
- **Editor de Texto/IDE:**
 - **VS Code (Visual Studio Code):** Um editor popular com excelentes extensões para C/C++ (incluindo depuração).
 - **Editores de Texto Simples:** nano, vim ou emacs diretamente no terminal Linux.

- **5. Requisitos e Entregáveis**
- **5.1. Funcionalidades Mínimas:** O simulador deve incluir as operações listadas nos Objetivos Específicos 3.1, 3.2, 3.3 . O Objetivo 3.4 (Simulação de Alocação de Blocos).
- **5.2. Interface:** O simulador deve ser operado via linha de comando, oferecendo um menu interativo ou aceitando argumentos para as operações.
- **5.3. Tratamento de Erros:** O programa deve ter estratégias para tratar erros comuns (ex: arquivo/diretório não encontrado na simulação, permissão negada (eventuais arquivos a serem excluídos sem permissão, nome inválido) com mensagens claras.
 - **5.4. Código-Fonte:** Trechos do código-fonte relevantes devem ser bem comentados, organizado e seguir boas práticas de programação.
- **5.5. Documentação (README.md):** Um arquivo README.md detalhado no repositório do projeto, contendo:
 - Instruções de como compilar e executar o simulador.
 - Explicação das escolhas de design e das estruturas de dados utilizadas (e.g., como structs e ponteiros foram usados para representar FCBs e a árvore de diretórios).
 - **Discussão explícita de como o trabalho implementa e demonstra os conceitos teóricos da disciplina**, como:
 - Conceito de arquivo e seus atributos.
 - Operações com arquivos (criar, ler, escrever, excluir, etc.).
 - A representação do File Control Block (FCB) e o "inode" simulado.
 - A estrutura de diretórios em árvore e suas vantagens (eficiência, nomeação, agrupamento).
 - O mecanismo de proteção de acesso (RWX, proprietário/grupo/outros) e a implementação de chmod com bitmasks.
 - Simulação da alocação de blocos (contínua, encadeada, indexada).
 - Exemplos de uso do simulador e comparação com comandos Linux reais.

6. Critérios de Avaliação

- **Funcionalidade e Robustez (30%):**
 - Todas as operações propostas funcionam corretamente.
 - Robustez no tratamento de erros.
 - Gerenciamento adequado da memória (evitar *memory leaks*).
- **Fidelidade aos Conceitos Teóricos (10%):**
 - Clareza e precisão na implementação das estruturas de dados (FCB, diretórios).
 - Efetividade do controle de acesso.
 - Conexão explícita entre a implementação e os conceitos teóricos na documentação.
 - Valor agregado de funcionalidades bônus (links, alocação de blocos).
- **Qualidade e Organização do Código (5%):**
 - Código limpo, modular, com comentários relevantes.
 - Uso adequado da linguagem C (ponteiros, alocação dinâmica, typedef, enums).
- **Apresentação (50%):**
 - Foco na explicação dos conceitos teóricos e da relação com a implementação.
- **Trabalho pode ser feito em dupla ou trio.**
- **Data da entrega: 08/12 (segunda-feira) até as 23:59.**