Pontifícia Universidade Católica de Minas Gerais
Departamento de Ciência da Computação
Disciplina: Sistemas Operacionais - TP
Professor: Pedro Ramos

VALOR: 20 PONTOS (10 código + 10 entrevista)
ENTREGA: 20/06/2025

Sistema de Arquivos

POSIX (Portable Operating System Interface), que pode ser traduzido como Interface Portável de Sistema Operacional) é uma família de normas definidas pelo IEEE para a manutenção de compatibilidade entre sistemas operacionais. POSIX define uma interface de programação de aplicações (API), juntamente com *shells* de linha de comando e interfaces utilitárias, para compatibilidade de software com variantes de Unix e outros sistemas operacionais. [1]

Neste trabalho, vocês implementarão um sistema de arquivos virtual em Java. Para tal, vamos simular o comportamento de **algumas chamadas POSIX**.

Introdução

Em sistemas Linux, o sistema de diretórios é organizado em uma árvore a partir da raiz "/". Por exemplo, se quisermos acessar os binários de um usuário, precisamos ir até "/usr/local/bin". Veja a imagem abaixo.

Os diretórios são "montados"como uma visita em profundidade na árvore a partir da raiz "/"até chegar na folha desejada. A folha pode ser um arquivo ou um diretório vazio.

Todo arquivo é também um diretório, mas nem todo diretório é um arquivo.

Também será necessário lidar com arquivos grandes. O que fazer quando o tamanho do arquivo excede o tamanho do *buffer* disponível na memória?

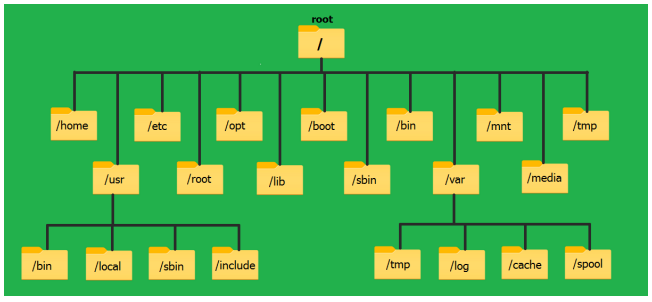

Figura 1: Estrutura de diretórios Linux

Especificação do trabalho

Vocês deverão implementar um sistema de arquivos virtual que suporte as seguintes operações POSIX:

- `mkdir` (criação de diretórios)

  Como citado anteriormente, os diretórios são organizados em uma estrutura de dados no formato de uma **árvore**, em que a raiz é o diretório raiz (**root**).

- `touch` (criação de arquivos vazios)

  O comando *touch* cria um arquivo vazio. Considere organizar o arquivo em blocos de bytes: o que pode acontecer se você tentar escrever um arquivo com mais de 4GB em um array de bytes?

  Porquê optar por um armazenamento em blocos de bytes (uma indireção)? Re: Para controlar o **fluxo de dados** entre memória e SSD/disco, e fazer uso do coletor de lixo do Java.

- `ls` (lista arquivos e subdiretórios)

  O comando `ls` lista todo o conteúdo de um diretório. Pode ser executado recursivamente, ou seja, imprimir todo o conteúdo do diretório e dos sub diretórios e arquivos.

- `cp` (cópia de arquivos)

  O comando `cp` copia o conteúdo de um arquivo para outro, ou o conteúdo de um diretório para outro. Também pode executar recursivamente e copiar todos os subdiretórios e arquivos.

- `mv` (movimentação e renomeação de arquivos e diretórios)

  Move um arquivo ou diretório de um lugar pra outro. É naturalmente recursivo.

- `write` (escrita em arquivos)

  Esta escrita é sequencial, ou seja, a partir de um determinado *offset*, escreve-se todo o conteudo do buffer de escrita em um arquivo.

  Caso desejar, o usuario poderá anexar um novo dado ao final do arquivo com a opção `append`.

  Caso o buffer de escrita não seja grande o suficiente para escrever todo o conteúdo no arquivo alvo, o mesmo deve ser dividido em *chunks* ou "blocos".

- `read` (leitura de arquivos)

  A leitura também é sequencial a partir do início do arquivo. Lê-se todo o conteúdo do arquivo e armazena-se em um *buffer*.

  Se o *buffer* não for grande o suficiente para ler todo o conteúdo do arquivo, pode-se realizar a leitura por partes.

- `rm` (remoção de arquivos e diretórios)

  Permanentemente deleta/remove qualquer diretório ou arquivo. Apenas usuários com permissão de escrita (`w`) podem chamar o `rm` dentro de determinado diretório.

  O comando é opcionalmente recursivo, ou seja, pode apagar recursivamente todos os itens contidos dentro de um diretório.

  Caso o diretório tenha outros subdiretórios, o uso do `rm` recursivo é obrigatório.

- `chmod` (configuração da permissão de arquivo ou diretório)

  Todas as chamadas de sistema necessitam de **permissão** para serem executadas.

  O usuário raiz (**root** ou também conhecido como **kernel**) tem permissão para ler (R), escrever (W) e executar (X) em qualquer diretório ou arquivo.

  Usuários com permissão de **root** também tem os mesmo privilégios.

  Do contrário, usuários só podem realizar operações de leitura (R), escrita (W) ou execução (X) em arquivos ou diretórios que tenha criado.

  É possível configurar a permissão de um usuário da seguinte forma com a chamada **chmod**.

  Por exemplo, um usuário pode ter permissão completa (`r w x`) ou nenhuma (- - -) para acesso a um determinado arquivo ou diretório.

Uma sugestão de implementação:

```
public class MetaDados {
    private String nome;
    private int tamanho;
    private String dono;

    // <"user", "rwx">
    private HashMap<String, String> permissoes;
}

public class Bloco {
    private Byte[] dados;
}

public class Arquivo {
    private MetaDados metaDados;
    private Bloco[] arquivo;
}

public class Diretorio {
    private MetaDados metaDados;
    // Filhos do diretorio
    private Arquivo[] arquivos;
    private Diretorio[] subDirs;
}

public class FileSys {
    private Diretorio raiz;
}
```

O código acima é uma mera sugestão.

No código base disponibilizado pelo professor, não há implementação do Sistema de Arquivos. Há uma interface chamada `IFileSysten` que deve ser implementada por você sem modificá-la drasticamente, exceto em caso de necessidade (justificada).

Os testes junit também serão adicionados a esse repositório base. Lembre-se de ficar atento para realizar um `git rebase` quando os testes forem *commitados*.

Dê uma olhada:

Código base: LINK REPOSITÓRIO BASE

Requisitos do trabalho.

- Entrega do código compilável e menu interativo funcional. O professor realizará testes simples a partir do menu interativo (presente na classe `Main`).

  Código + Avaliação breve do professor via menu interativo => 5 pontos

- Testes Junit. Você deve implementar testes de cobertura para todas as chamadas. Estes testes corresponderão a 5 pontos da nota, e o repositório possui um *.jar* do junit.

  Testes Junit => 5 pontos

- Entrevista sobre o trabalho com todos os integrantes do grupo. O professor sorteará perguntas sobre Sistemas de Arquivos e sobre a implementação do projeto, bem como as dificuldades, obstáculos enfrentados, como foi a dinâmica no grupo, etc.

  Entrevista de TP Presencial => 10 pontos

Os grupos podem conter 1, 2 ou 3 pessoas.

O que deve ser entregue:

Você deve:

1. Implementar a interface `IFileSystem` presente no código base sem modificá-la, a não ser em caso de necessidade extrema.
2. Entregar o link para um repositório público no github contendo a solução da implementação. Somente commits até a data de entrega serão avaliados.
3. Abrir um PR (pull request) do seu repositório para o repositório base (a partir de um *fork*, por exemplo).
4. Se preparar para a **entrevista sobre o trabalho**. As entrevistas ocorrerão nas datas 24/06/2025 e 27/06/2025. A ordem dos grupos ainda será definida.

Referências

[1] POSIX FAQ https://www.opengroup.org/austin/papers/posix$_faq.html$

Bom trabalho!