Universidade do Vale do Itajaí
Centro: Escola Politécnica
Curso: Ciência da Computação, Engenharia de Computação
Disciplina: Sistemas Operacionais

Sistemas de arquivos

Nome: Pedro Henrique Kons, Guilherme Thomy

30/06/2025

1. Resumo

Este relatório detalha o projeto e a implementação de um sistema de arquivos virtual em memória, utilizando uma estrutura de dados de Árvore B para gerenciar a hierarquia de diretórios e arquivos. O sistema suporta operações fundamentais como criação, remoção e navegação de diretórios, além de inserção e exclusão de arquivos de texto. O objetivo principal é demonstrar a aplicação de estruturas de dados avançadas na resolução de problemas clássicos de sistemas operacionais, como a organização de dados em disco.

2. Contexto e Explicações

Um sistema de arquivos é uma das abstrações mais fundamentais de um sistema operacional. Ele organiza dados em uma estrutura hierárquica, geralmente de diretórios e arquivos, permitindo armazenamento, busca e recuperação eficientes. Para simular essa estrutura em um ambiente virtual, a escolha da estrutura de dados subjacente é crucial para a performance e escalabilidade do sistema.

A Árvore B foi escolhida como a estrutura de dados central deste projeto por suas características intrínsecas, que a tornam ideal para sistemas de arquivos:

a) **Balanceamento Automático:** A árvore B se mantém balanceada após inserções e exclusões. Isso garante que a altura da árvore cresça de forma logarítmica em relação ao número de itens, mantendo as operações de busca, inserção e remoção com complexidade de tempo O(log n), mesmo para um número muito grande de arquivos em um diretório.
b) **Alta Ordem (Fanout):** Diferente de árvores binárias, cada nó de uma árvore B pode ter um grande número de filhos. Isso resulta em uma árvore mais "achatada", reduzindo o número de nós que precisam ser percorridos em uma busca, o que é especialmente vantajoso em sistemas de armazenamento em disco.
c) **Ordenação:** Os itens (arquivos e diretórios) em cada nó são mantidos em ordem alfabética, o que permite listar o conteúdo de um diretório de forma ordenada e eficiente.

Nesta implementação, cada diretório do sistema de arquivos contém sua própria instância de uma Árvore B. As "chaves" dessa árvore não são valores simples, mas sim ponteiros para uma estrutura TreeNode, que armazena metadados como nome, tipo (arquivo ou diretório) e um ponteiro para os dados reais (o conteúdo do arquivo ou a árvore B do subdiretório).

4. Códigos Relevantes de Implementação

A seguir serão apresentados os trechos de código mais importante que definem a arquitetura do sistema

4.1 Estruturas de dados (filesystem.h)

As estruturas abaixo definem a composição dos nós da árvore, dos arquivos e dos diretórios.

```c
// Define o grau mínimo da árvore. Influencia o número de chaves por nó.
#define BTREE_ORDER 3

// Enum para diferenciar arquivos de diretórios
typedef enum { FILE_TYPE, DIRECTORY_TYPE } NodeType;

// Nó da árvore, representa um arquivo ou diretório
typedef struct TreeNode {
    char* name;
    NodeType type;
    union {
        File* file;
        struct Directory* directory;
    } data;
} TreeNode;

// Nó da Árvore B
typedef struct BTreeNode {
    int num_keys;
    TreeNode* keys[2 * BTREE_ORDER - 1];
    struct BTreeNode* children[2 * BTREE_ORDER];
    int leaf; // Flag que indica se o nó é uma folha
} BTreeNode;

// Estrutura do diretório, que contém uma Árvore B
typedef struct Directory {
    BTree* tree;
} Directory;
```

4.2 Inserção na Árvore B (filesystem.c)

A função de inserção é o coração da Árvore B. Ela garante o balanceamento através da operação de split (divisão de nó) quando um nó está cheio.

```c
void btree_insert(BTree* tree, TreeNode* node) {
    if (!tree || !node) return;
    BTreeNode *r = tree->root;

    // Se a raiz está cheia, a árvore cresce em altura
    if (r->num_keys == 2 * BTREE_ORDER - 1) {
        BTreeNode *s = btree_create_node(0 /* não é folha */);
        tree->root = s;
        s->children[0] = r;
        btree_split_child(s, 0); // Divide a raiz antiga
        btree_insert_nonfull(s, node); // Insere no novo nó
    } else {
        btree_insert_nonfull(r, node);
    }
}
```

4.3 Remoção na Árvore B (filesystem.c)

A remoção em uma árvore B pode ser complexa. Para este projeto, foi implementada uma versão simplificada que remove itens (arquivos ou diretórios vazios) apenas de nós folha. Isso atende aos requisitos do projeto, que estipulam a remoção de diretórios apenas se estiverem vazios.

```c
// Função auxiliar que remove um item de um nó folha
static void btree_remove_from_leaf(BTreeNode *x, int idx) {
    for (int i = idx + 1; i < x->num_keys; ++i)
        x->keys[i - 1] = x->keys[i];
    x->num_keys -= 1;
}

// Função principal de deleção
void btree_delete(BTree* tree, const char* name) {
    if (!tree) return;
    // Btree_delete_node é uma função auxiliar recursiva que encontra
    // e remove o item, retornando 1 em caso de sucesso.
    int removed = btree_delete_node(tree->root, name);

    if (!removed)
        printf("Item '%s' não encontrado para remoção.\n", name);

    // Se a raiz ficar vazia após a remoção, a altura da árvore diminui
    if (tree->root->num_keys == 0 && !tree->root->leaf) {
        BTreeNode *old_root = tree->root;
        tree->root = old_root->children[0];
        free(old_root);
    }
}
```

4.4 Terminal Interativo (main_fs.c)

O loop principal do programa interpreta os comandos do usuário, criando uma interface de linha de comando para interagir com o sistema de arquivos.

```c
int main() {
    // ... inicialização ...

    char input[256];
    char command[64], arg1[128], arg2[512];

    while (1) {
        printf("%s$ ", current_path); // Exibe o prompt (ex: /Documentos$)
        fflush(stdout);

        if (!fgets(input, sizeof(input), stdin)) break;
        input[strcspn(input, "\n")] = 0; // Remove a quebra de linha

        // Parse simples dos comandos
        sscanf(input, "%s %s %[^\n]", command, arg1, arg2);

        // Processamento de comandos (mkdir, ls, cd, rm, etc.)
        if (strcmp(command, "mkdir") == 0) {
            // ... lógica para criar diretório ...
        } else if (strcmp(command, "ls") == 0) {
            list_directory_contents(current_dir);
        } else if (strcmp(command, "cd") == 0) {
            // ... lógica para navegar entre diretórios ...
        } else if (strcmp(command, "exit") == 0) {
            break;
        }
        // ... outros comandos ...
    }
```

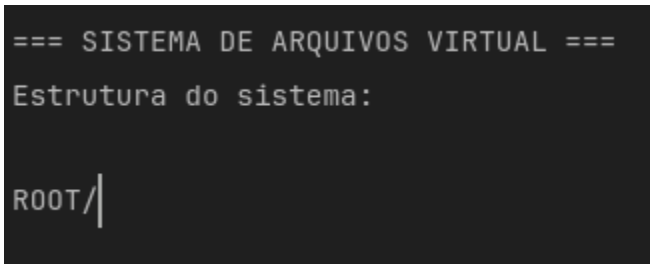
5. Resultados da simulação

Para validar a implementação, foi realizada uma simulação de uso do sistema de arquivos. A tabela a seguir mostra os comandos executados e as saídas correspondentes.

Comando Executado	**Saída do Sistema / Descrição**
./fs	Executado!
/$ ls	Conteúdo de /: (diretório vazio)
/$ mkdir Pasta	Diretório 'Pasta' criado com sucesso
/$ mkdir Arquivos	Diretório 'Arquivos' criado com sucesso.
/$ rmdir Projetos	Diretório 'Projetos' removido com sucesso.
/$ cd Documentos	Navegou para /Documentos
/Documentos$ echo relat.txt "Teste"	Arquivo 'relat.txt' criado com conteúdo.
/Documentos$ rm relat.txt	Arquivo 'relat.txt' removido com sucesso.
Documentos$ cd ..	Voltou para /
/$ save	Sistema de arquivos salvo em fs.img

Conteúdo do Arquivo de Persistência fs.img

Após a execução dos comandos, o arquivo fs.img refletiu o estado final do sistema, com todos os itens criados e depois removidos, validando o ciclo completo de operações.



6. Conclusão

O desenvolvimento deste sistema de arquivos virtual demonstrou com sucesso a eficácia da Árvore B como estrutura de dados para o gerenciamento de sistemas de arquivos hierárquicos. Todos os requisitos funcionais propostos, incluindo criação, remoção, navegação e listagem, foram implementados e validados através de simulações.

A implementação atual serve como uma base sólida, podendo ser estendida com funcionalidades mais avançadas no futuro, como suporte a permissões de usuário, timestamps de modificação, e uma implementação mais robusta de caminhos relativos e absolutos. O projeto foi fundamental para solidificar os conhecimentos teóricos de estruturas de dados e sua aplicação prática em problemas de sistemas operacionais.