# Relatório do Trabalho Prático: DCC605 File System Shell (DCC-FSSHELL)

**Disciplina:** Sistemas Operacionais
**Equipe:**

Giovana Piorino Vieira de Carvalho (Matrícula: 2022035989)

Gustavo Dias Apolinário (Matrícula: 2022035911)

Matheus Galdino Ferreira (Matrícula: 2022035776)

---

**Nota:** Este trabalho foi feito em **trio**. Originalmente a dupla era Giovana Piorino Vieira de Carvalho e Matheus Galdino Ferreira. A dupla do aluno Gustavo Dias Apolinário era o Marcos Lorran Bitencourt Aguilar, assim como no Trabalho Prático 2. Contudo, Marcos teve problemas pessoais e não pode realizar esse trabalho, sua desistência foi **avisada muito em cima da hora**. Dado o tamanho do Trabalho Prático 3 não era possível terminá-lo sozinho com o tempo restante. Com isso, foi feita uma colaboração de última hora com a dupla já pronta. **Todos os três colaboraram igualmente** e foram utilizadas ideias dos códigos já realizados pela dupla e pelo terceiro membro, sendo feita uma fusão e uma análise muito interessante das diferenças gerando um novo código que foi o resultado final. Peço uma compreensão especial, dado a **situação atípica** somada a data do Trabalho Prático 3, sendo bem perto do final do semestre, dificultando o encontro de uma nova dupla.

---

## 1. Introdução

Este trabalho consistiu na implementação de um shell interativo para manipulação de sistemas de arquivos no formato **ext2**, baseado no *Berkley Fast File System (FFS)*. O objetivo foi desenvolver um conjunto de comandos que permitem navegar e analisar a estrutura interna de uma imagem de disco ext2, incluindo operações como listagem de arquivos, navegação entre diretórios, visualização de metadados e exploração hierárquica do sistema de arquivos. O projeto foi desenvolvido em linguagem C, utilizando estruturas de dados definidas pelo sistema ext2 e acesso direto a blocos de disco.

---

## 2. Desenvolvimento

### 2.1. Estruturas de Dados e Leitura Inicial

O sistema ext2 organiza os dados em blocos, grupos e inodes. Para interpretar a imagem, foram utilizadas as seguintes estruturas:

- **Superbloco (ext2_super_block)**: Contém metadados globais do sistema de arquivos, como contagem de blocos, inodes livres e tamanho do bloco.
- **Descritores de Grupo (ext2_group_desc)**: Armazenam informações sobre os grupos de blocos, incluindo a localização dos bitmaps e da tabela de inodes.
- **Inodes (ext2_inode)**: Representam os metadados de arquivos e diretórios, como permissões, tamanho e ponteiros para blocos de dados.
- **Entradas de Diretório (ext2_dir_entry)**: Mapeiam nomes de arquivos para inodes.

**A leitura inicial do sistema de arquivos é realizada em três etapas:**

1. **Leitura do Superbloco**:

```c
void read_superblock(int fd) {
    superblock = malloc(sizeof(struct ext2_super_block));
    if (!superblock) {
        perror("Erro ao alocar memória para superbloco");
        return;
    }

    if (lseek(fd, BASE_OFFSET, SEEK_SET) == -1) {
        perror("Erro ao posicionar ponteiro no superbloco");
        free(superblock);
        return;
    }

    ssize_t bytes_read = read(fd, superblock, sizeof(struct ext2_super_block));
    if (bytes_read != sizeof(struct ext2_super_block)) {
        perror("Erro ao ler superbloco");
        free(superblock);
        return;
    }
}
```

2. **Leitura dos Descritores de Grupo**:

```c
void read_blockgroup(int fd) {
    int num_groups = (superblock->s_blocks_count - 1) / superblock->s_blocks_per_group + 1;
    size_t group_desc_size = sizeof(struct ext2_group_desc) * num_groups;

    blockgroup = malloc(group_desc_size);
    if (!blockgroup) {
        perror("Erro ao alocar memória para grupos de blocos");
        return;
    }

    off_t blockgroup_offset = BASE_OFFSET + (BLOCK_SIZE << superblock->s_log_block_size);
    if (lseek(fd, blockgroup_offset, SEEK_SET) == -1) {
        perror("Erro ao posicionar ponteiro nos grupos de blocos");
        free(blockgroup);
        return;
    }

    ssize_t bytes_read = read(fd, blockgroup, group_desc_size);
    if (bytes_read != group_desc_size) {
        perror("Erro ao ler grupos de blocos");
        free(blockgroup);
        return;
    }
}
```

3. **Leitura da Tabela de Inodes**:

```c
void read_inodeTable(int fd) {
    if (!superblock || !blockgroup) {
        printf("[ERRO] Superblock ou blockgroup não foram carregados.\n");
        return;
    }

    __le32 inodes_per_group = superblock->s_inodes_per_group;
    __le16 inode_size = superblock->s_inode_size;

    size_t table_size = inodes_per_group * inode_size;

    inodes = malloc(table_size);
    if (!inodes) {
        perror("[ERRO] Erro ao alocar memória para a tabela de inodes");
        return;
    }

    off_t inode_table_offset = BLOCK_OFFSET(blockgroup->bg_inode_table);

    if (lseek(fd, inode_table_offset, SEEK_SET) == -1) {
        perror("[ERRO] Erro ao posicionar ponteiro na tabela de inodes");
        free(inodes);
        return;
    }

    ssize_t bytes_read = read(fd, inodes, table_size);
    if (bytes_read != table_size) {
        perror("[ERRO] Erro ao ler a tabela de inodes");
        free(inodes);
        return;
    }

    printf("Tabela de inodes carregada com sucesso.\n");
}
```

---

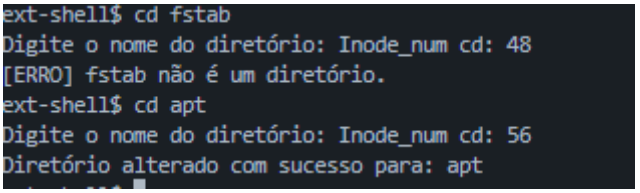
### 2.2. Implementação dos Comandos

#### 2.2.1. ls (Listar Diretório)

O comando ls tem como objetivo listar os arquivos e diretórios presentes no diretório atual. Para isso, ele começa obtendo o inode do diretório atual, que é armazenado na variável **current_directory_inode**. Esse inode contém os ponteiros para os blocos de dados onde as entradas do diretório estão armazenadas. Cada bloco de dados é percorrido sequencialmente, e as entradas do diretório (**ext2_dir_entry**) são lidas. O campo file_type de cada entrada indica se é um arquivo regular (**EXT2_FT_REG_FILE**), um diretório (**EXT2_FT_DIR**) ou outro tipo de arquivo. Para cada entrada válida (ignorando as entradas . e ..), o nome do arquivo/diretório e seu inode são exibidos. Diretórios são marcados com uma barra (/) ao final do nome, facilitando a identificação visual.

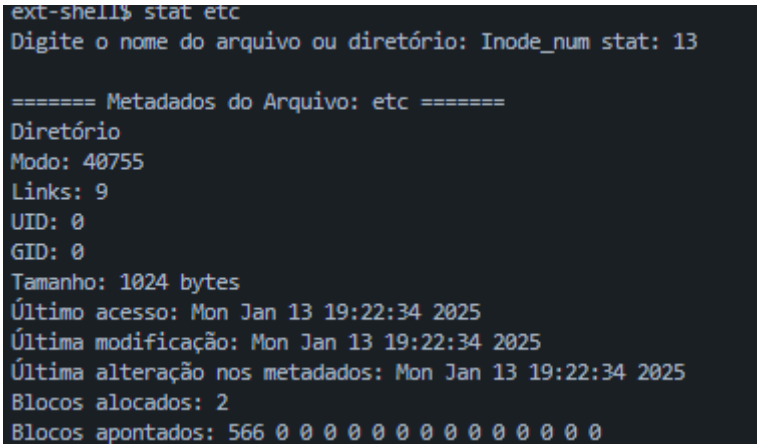

#### 2.2.2. cd (Mudar Diretório)

O comando cd permite a navegação entre diretórios. Quando o usuário deseja entrar em um diretório específico, o nome do diretório é passado como argumento. O sistema busca o inode correspondente ao diretório usando a função **findInodeByName**. Caso o inode encontrado não seja um diretório (**file_type != EXT2_FT_DIR**), o sistema informa que não é possível navegar para um arquivo. Para navegar para o diretório pai (..), o inode do diretório pai é obtido da entrada .. no diretório atual. Se o diretório atual for a raiz (**EXT2_ROOT_INODE**), o sistema informa que já está no diretório raiz. Após a validação, o current_directory_inode é atualizado para o inode do novo diretório, permitindo que os comandos subsequentes operem no contexto correto.



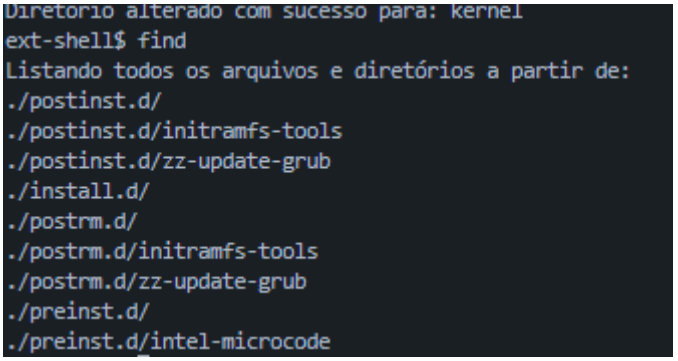
### 2.2.3. stat (Metadados)

O comando stat exibe os metadados de um arquivo ou diretório. Ele começa buscando o inode correspondente ao nome do arquivo ou diretório fornecido, utilizando a função **findInodeByName**. Uma vez obtido o inode, os metadados são lidos e formatados para exibição. Esses metadados incluem o tamanho do arquivo, as datas de acesso, modificação e criação, as permissões, o número de blocos alocados e os ponteiros para os blocos de dados. As datas são convertidas para um formato legível usando a função ctime, e os blocos apontados pelo inode são listados para fornecer uma visão detalhada da alocação de dados.



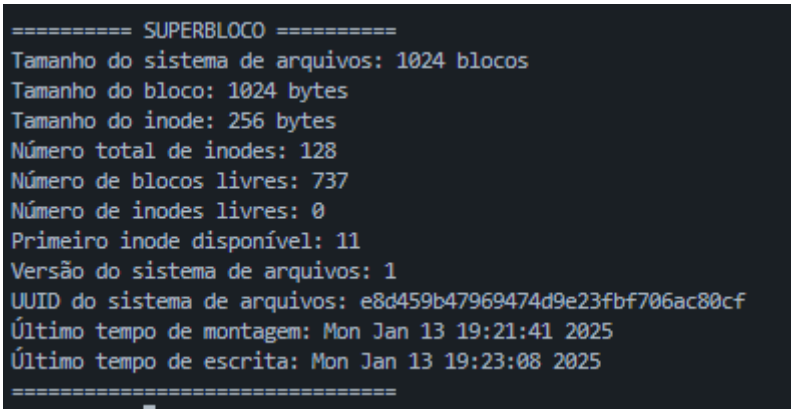

### 2.2.4. find (Explorar Hierarquia)

O comando find realiza uma busca recursiva a partir do diretório atual, listando todos os arquivos e subdiretórios. Ele começa lendo as entradas do diretório atual, ignorando as entradas . e .. .Para cada entrada válida, o caminho completo é construído e exibido. Se a entrada for um diretório (**EXT2_FT_DIR**), a função findFile é chamada recursivamente, passando o caminho atualizado como argumento. Esse processo continua até que todos os arquivos e subdiretórios tenham sido listados, proporcionando uma visão hierárquica completa do sistema de arquivos a partir do diretório atual.



### 2.2.5 sb (Superbloco)

O comando sb exibe as informações contidas no superbloco do sistema de arquivos. O superbloco é uma estrutura crítica que contém metadados globais, como o número total de blocos e inodes, o tamanho do bloco, o número de blocos e inodes livres, o UUID do sistema de arquivos e os tempos de montagem e escrita. Para exibir essas informações, o sistema lê o superbloco diretamente do início da imagem do disco, utilizando o offset **BASE_OFFSET**. Os dados são então formatados e exibidos de maneira clara, incluindo informações como o tamanho do sistema de arquivos, o número de grupos de blocos, o primeiro inode disponível e o UUID. Datas importantes, como o último tempo de montagem e escrita, são convertidas para um formato legível usando a função ctime.




---

## 3. Conclusão

O trabalho demonstra a complexidade de sistemas de arquivos modernos, permitindo a exploração prática de conceitos como inodes, diretórios e alocação de blocos. A implementação reforçou o entendimento do layout do ext2 e a importância de otimizações como grupos de blocos.