Universidade Federal
de São João del-Rei

Implementação de um Simulador de Sistema de Arquivo

Sistemas Operacionais

Professor: Rafael Sachetto Oliveira

Pedro Garcia

Matheus Tavares Elias

1. INTRODUÇÃO

Este trabalho consiste na implementação de um simulador de Sistema de Arquivo em linguagem C. Tem como objetivo aprofundar o entendimento sobre sistemas de arquivos em um sistema operacional. Ele consiste na implementação de um simulador de um sistema de arquivos simples, baseado em uma tabela de alocação de 16 bits (FAT), juntamente com um shell para realizar operações nesse sistema de arquivos.

O sistema de arquivos virtual será armazenado em uma partição virtual, e todas as estruturas de dados relacionadas serão mantidas em um único arquivo chamado "fat.part". A partição virtual terá um tamanho total de 4MB, com algumas configurações iniciais padrão de setor e cluster.

A estrutura do sistema de arquivos consistirá em um bloco de inicialização (boot block), seguido da FAT (File Allocation Table), do diretório raiz e, por fim, da seção de dados contendo os clusters restantes.

A FAT é responsável por mapear a estrutura dos blocos da partição. Neste sistema de arquivos, terá um tamanho de 8 clusters (8192 bytes), permitindo o armazenamento de 4096 entradas de blocos. Após a FAT, encontra-se o diretório raiz, que também possui o tamanho de 1 cluster. O diretório raiz contém um conjunto de entradas de diretório que podem apontar para outros diretórios ou arquivos. Inicialmente, todas as entradas de diretório estarão livres, e todas as estruturas serão inicializadas com o valor 0x00. Após a FAT e o diretório raiz, temos a seção de dados, que contém o restante dos clusters. Cada cluster tem o tamanho de 1024 bytes e será usado para armazenar dados de arquivos.

O shell desenvolvido como parte deste trabalho permite executar operações no sistema de arquivos simulado. No desenvolvimento deste simulador, serão utilizadas diferentes rotinas, incluindo funções como init(), load(), ls(), mkdir(), create(),unlink(), append(), read() e write(), além do auxiliar help() , cada uma responsável por realizar uma operação específica no sistema de arquivos.

Espera-se que, por meio deste trabalho, seja aprimorado o conhecimento sobre sistemas de arquivos, a implementação de estruturas de dados relacionadas e a interação com um sistema operacional simulado.

Para rodar o programa basta compilar com o utilitário make e rodar com ./fatsim. A partir daí, basta digitar os comandos um por um no shell. Os diretórios sempre começam com o /, por exemplo, se quiser listar os diretórios no root basta digitar “ ls / ” e para criar um diretório no root, “mkdir /teste” por exemplo.

2. LISTAGEM DAS ROTINAS

Segue abaixo a listagem das rotinas utilizadas:

2.1. init():

Inicializa o sistema de arquivos. Cria um bloco de inicialização, inicializa a FAT (Tabela de Alocação de Arquivos) e o bloco do diretório raiz, e salva essas informações em um arquivo.

2.2. load():

Carrega o sistema de arquivos a partir de um arquivo chamado "fat.part". Lê o bloco de inicialização e a FAT do arquivo e verifica a integridade do bloco de inicialização.

2.3. ls(char* directories):

Lista o conteúdo de um diretório especificado pelo parâmetro *directories*. Separa o caminho em diretórios individuais, começando pelo diretório raiz. Percorre os diretórios no caminho e verifica se cada diretório existe no diretório anterior. Por fim, imprime os nomes de arquivo de todos os arquivos/diretórios no diretório especificado.

2.4. mkdir(char* directories):

Cria um novo diretório com o nome especificado no caminho especificado. Separa o caminho em diretórios individuais, começando pelo diretório raiz. Percorre os diretórios no caminho e verifica se cada diretório existe no diretório anterior. Se o

caminho for válido e houver uma entrada livre no diretório pai, cria uma nova entrada de diretório e atualiza a FAT e os blocos de diretório conforme necessário.

2.5. create(char* directories):

Cria um novo arquivo com o nome especificado no caminho especificado. Segue uma lógica similar à função *mkdir()*, mas, em vez de criar uma entrada de diretório, cria uma entrada de arquivo no diretório pai.

2.6. unlink(char* directories):

Remove um arquivo ou diretório com o nome especificado no caminho especificado. Segue uma lógica similar à função *mkdir() e create()*, mas, em vez de criar uma nova entrada, busca a entrada com o nome especificado e a remove do bloco de diretório. Para diretórios, também verifica se o diretório está vazio antes de removê-lo e desaloca as entradas correspondentes na FAT.

2.7. append(char* filename, char* data):

Anexa os dados especificados ao arquivo com o nome especificado. Abre o arquivo, posiciona o ponteiro no final do arquivo e escreve os dados. Atualiza os blocos de dados e a FAT conforme necessário.

2.8. read(char* filename):

Lê o conteúdo do arquivo com o nome especificado e o retorna como uma string. Abre o arquivo, lê os dados e retorna o conteúdo.

2.9. write(char* filename, char* data):

Escreve os dados especificados no arquivo com o nome especificado, substituindo o conteúdo existente. Abre o arquivo, posiciona o ponteiro no início do arquivo, escreve os dados e atualiza os blocos de dados e a FAT conforme necessário.

3. ESTRUTURA DO CÓDIGO

A estrutura do código é composta por dois arquivos: main.c e fat.c, que contém as implementações principais.

3.1. MAIN

A função main começa limpando a tela do console usando o comando "system("clear")". Em seguida, declara uma variável `input_str` como uma matriz de caracteres, que será usada para armazenar os comandos inseridos pelo usuário. Em um loop infinito (`while(1)`), o código lê os comandos do usuário usando a função `fgetc(stdin)` e armazena-os na matriz `input_str`. O loop continua até que o usuário pressione a tecla Enter.

O comando inserido é então comparado com uma série de strings usando a função `strcmp()` para determinar qual operação deve ser executada. Se o comando corresponder a "init", "load", "exit", "clear", a função correspondente é chamada (`init()`, `load()`, `exit()`, `system("clear")).

Se o comando contiver aspas duplas, indica-se que é uma operação de escrita (`write`) ou de anexação (`append`). Os argumentos relevantes (a string e o caminho) são extraídos do comando usando a função `strtok()` e passados para as funções `write()` ou `append()`.

Se o comando não corresponder a nenhuma das opções anteriores, assume-se que é uma das operações básicas do sistema de arquivos, como `ls`, `mkdir`, `create`, `unlink` ou `read`. Os argumentos relevantes são extraídos usando a função `strtok()` e passados para as funções correspondentes (`ls()`, `mkdir()`, `create()`, `unlink()`, `read()`). Caso o comando não corresponda a nenhuma operação conhecida, é exibida uma mensagem de "Comando não encontrado!".

Em resumo, a função `main()` implementa a lógica principal do shell do sistema de arquivos simulado, permitindo a interação do usuário e a execução das operações disponíveis. É responsável por receber os comandos, analisá-los e chamar as funções correspondentes para manipular o sistema de arquivos simulado.

3.2. FAT

O arquivo `FAT.C` contém a implementação de várias funções auxiliares e funções utilizadas no shell do sistema de arquivos simulado. Essas funções são responsáveis por realizar operações relacionadas à leitura e gravação de dados no sistema de arquivos, gerenciamento da tabela FAT (File Allocation Table) e manipulação de diretórios e arquivos.

As funções auxiliares incluem:

3.2.1. __readCluster__(int index):

Lê um cluster de dados do sistema de arquivos a partir de um índice fornecido e retorna o cluster lido.

3.2.2. __writeCluster__(int index, union data_cluster *cluster):

Grava um cluster de dados no sistema de arquivos usando o índice fornecido e o cluster fornecido.

3.2.3. __findFreeSpaceFat__():

Procura por um espaço livre na tabela FAT e retorna o índice do primeiro espaço livre encontrado.

3.2.4. __writeFat__():

Grava a tabela FAT no sistema de arquivos.

3.2.5. __slice_str__(char * str, char * buffer, int start, int end):

Extrai uma parte de uma string e a coloca em um buffer, com base nos índices de início e fim fornecidos.

3.2.6. __resize__(char* directories, size_t extend_size):

Redimensiona um diretório no sistema de arquivos, aumentando seu tamanho em extend_size bytes.

As funções utilizadas no shell incluem as que foram mencionadas anteriormente na listagem de rotinas (item 2), onde essas funções implementam as operações básicas do sistema de arquivos simulado, permitindo a manipulação e gerenciamento de diretórios e arquivos.

Em resumo, o arquivo `FAT.C` contém as funções que operam diretamente no sistema de arquivos simulado, permitindo a leitura, gravação, criação e exclusão de arquivos e diretórios, além de gerenciar a tabela FAT. Essas funções são chamadas pela função `main()` no arquivo `main.c` para executar as operações solicitadas pelo usuário no shell do sistema de arquivos.

4. CONCLUSÃO

A implementação de um simulador de sistema de arquivos em linguagem C. Através desse projeto, foi possível aprofundar o entendimento sobre sistemas de arquivos em sistemas operacionais, explorando a implementação de estruturas de dados e a interação com um sistema de arquivos simulado.

Durante a implementação, foram utilizadas diferentes rotinas e funções, como `init()`, `load()`, `ls()`, `mkdir()`, `create()`, `unlink()`, `write()`, `append()`, `read()` , cada uma responsável por realizar uma operação específica no sistema de arquivos simulado. Essas rotinas permitiram a criação, manipulação e exclusão de diretórios e arquivos, além da leitura e escrita de dados.

O simulador de sistema de arquivos desenvolvido seguiu uma estrutura de armazenamento com um bloco de inicialização, tabela de alocação de arquivos (FAT), diretório raiz e seção de dados. Limitações foram aplicadas para simplificar a implementação, como o tamanho máximo da FAT e o número máximo de entradas de diretório.

Ao concluir, foi possível aprimorar o conhecimento sobre sistemas de arquivos, tanto em relação às estruturas de dados utilizadas quanto à manipulação e interação com o sistema operacional simulado. Além disso, a implementação do shell permitiu a execução de diversas operações no sistema de arquivos. Em suma, o trabalho de implementação do simulador de sistema de arquivos foi interessante e enriquecedor, proporcionando uma compreensão mais profunda sobre o funcionamento dos sistemas de arquivos em sistemas operacionais.