Trabalho 2 Sistemas Operacionais

Chiara Paskulin, Marcio do Carmo, Octavio Carpes, Otavio Bonder

[1]Pontifícia Universidade Católica do Rio Grande do Sul

chiara.paskulin@acad.pucrs.br

marcio.carmo@acad.pucrs.br

octavio.carpes@acad.pucrs.br

otavio.bonder@acad.pucrs.br

Resumo. *Este trabalho contempla uma tarefa, passada pelo professor, sobre o conteúdo de sistemas de arquivo da cadeira de sistemas operacionais. Neste exercício devemos simular um sistema de arquivo baseado em uma FAT.*

1. Introdução

Com o objetivo de aprimorarmos os nossos conhecimentos em relação à disciplina e conteúdos apresentados até o momento, nos foi proposto criar um sistema de arquivos, baseado no tipo FAT(file allocation table), na linguagem de nossa escolha. O sistema deve realizar diversas ações como criar diretórios e arquivos, listar o conteúdo de um diretório e ler o conteúdo do arquivo. Para realizar esta tarefa optamos por criar este sistema utilizando a linguagem Java.

O sistema de arquivos deve permitir a interação com o usuário através de um ”Shell”, comandos de linha, e deve executar os seguintes comandos:

- **init** - Deve inicializar o sistema de arquivos com as estruturas de dados.
- **load** - Carregar o sistema de arquivos do disco
- **ls** - Listar o conteúdo de um diretório
- **mkdir** - Criar um diretório
- **create** - Criar um arquivo
- **unlink** - Excluir um arquivo ou diretório (o diretório precisa estar vazio)
- **write** *”string”* - Escrever dados em um arquivo
- **append** - Anexar dados em um arquivo
- **read** - ler o conteúdo de um arquivo

Além disso, o sistema deve ser armazenado em uma partição virtual e suas estruturas de dados devem ser mantidas em um arquivo chamado *filesystem.dat*, que representará a partição. O tamanho desta partição é determinado por blocos de 1024 bytes e 2048 blocos. Os primeiros 4 blocos do disco armazenarão a FAT e o 5 bloco será responsável por armazenar o diretório raiz (*root*).

2. Desenvolvimento

Para desenvolvermos o trabalho em conjunto, optamos por utilizar gitflow para maior organização do código fonte entre os integrantes do projeto. Durante o desenvolvimento também optamos por utilizar apenas a classe *FileSystem*, como principal, e implementar os métodos necessários para fazer com que o programa funcione. Além disso, também temos uma classe que define uma entrada de diretório, a classe *DirEntry*.

2.1. Shell

O método *shell()* é a função responsável por realizar a interface com o usuário. É neste método que chamamos as funções do sistema de arquivos. Basicamente a função consiste de um ”*while(true)*”que mostra espera o comando do usuário e realiza a função desejada caso o comando passado pelo usuário exista.

2.2. init

O método *init()* inicializa o sistema de arquivos com as estruturas de dados, semelhante a formatar o sistema de arquivos virtual. Este método inicializa a FAT, primeiros 4 blocos, inicializa o diretório root(5 bloco) e zera todos os outros blocos restantes.

2.3. ls

A função *ls(String path)* serve para listar os arquivos e diretórios contidos em um diretório. Ela recebe como parâmetro o caminho para o diretório o qual desejamos listar os arquivos, como pro exemplo *ls /diretorioA/arquivoB.txt*. Neste método é realizado um ”split(/)”no *path* para obtermos um array contendo todos os elementos do path e armazenamos este array em uma variável, que por sua vez é passada por parâmetro para a função *followUntilFindDir(String[] path, short blocoAtual)*.

O método *followUntilFindDir* acessa os subdiretórios até o último e chama o método *accessAndListDir* para listar o ultimo diretório. O método *accessAndListDir* basicamente escreve no terminal qual é o conteúdo do diretório.

2.4. mkdir

O método *mkdir(String path)* cria um diretório no caminho desejado. A função possui as seguintes instruções, primeiramente realizamos um ”*split(”/”)*”no caminho passado por parâmetro para que possamos utilizar os nomes dos arquivos, por exemplo, caso o caminho seja ”/a/b”a nossa variável irá receber o valor de [”a”, ”b”]. Após criado este array com o caminho, chamado de *arrOfStr*, nós criamos uma variável chamada *newPath* e passamos o path a partir do diretório seguinte através de um loop.

Finalmente, utilizamos o *newPath* para chamar o método privado *followUntilCreateDir(newPath, (short) 4)*. A função privada *followUntilCreateDir* acessa os subdiretórios até o último e chama outro método privado para criar o novo diretório. Além disso na função *followUntilCreateDir* temos uma verificação para validar se o caminho passado por parâmetro é válido, caso o caminho não seja válido é exibido na tela uma mensagem de erro informando que o diretório passado por parâmetro não existe.

2.5. create

A implementação do método *create* foi feita da seguinte forma, primeiramente nós temos 3 parâmetros em nossa versão de create, são eles *path, content* e *size*. A seguir realizamos um *split(”/”)* no caminho passado por parâmetro (variável *path*), para que armazenar os dados em um vetor de strings contendo o caminho final do arquivo a ser criado e então chamamos uma função privada *followUntilCreateArchive(stringArray, (short) 4, content, size)*. Caso o caminho especificado não exista, é mostrado na tela um aviso informando que o caminho é inválido.

Este método privado, por sua vez, é o responsável por chamar a função que ”caminha”na nossa estrutura de dados acessando os subdiretorios até o penúltimo item e chama *accessAndCreateArchive* para criar o arquivo. A função *accessAndCreateArchive*, então procura se o arquivo que desejamos criar já existe e, caso não, procuramos qual é a primeira entrada de diretório que está vazia e é criado o arquivo.

Quando um arquivo é criado, nós temos que validar se a FAT tem espaço o suficiente para armazenar os dados e, caso tenha, devemos realizar um tratamento especial para arquivos de tamanho grande, maiores do que 1024 bytes, no qual devemos calcular a quantidade de blocos necessários para podermos armazenar os dados do arquivo corretamente. Nesta ”montagem especial”, nós marcamos cada bloco com 0x7fff, indicando que o bloco está em uso, e o último bloco indicando final de arquivo. Após todas as operações de manipulação de blocos e operações na FAT, nós escrevemos os dados atualizados no arquivo *filesystem.dat* para atualizar os dados no disco.

2.6. read

A função read tem a responsabilidade de ler o conteúdo de um arquivo salvo em nosso sistema de arquivos. Sua implementação tem a seguinte forma, primeiramente chamamos o método *readArchive(String path)* e passamos o caminho para chegar no arquivo desejado por parâmetro e então realizamos um *split(”/”)* para armazenar o caminho em um array. Logo em seguida este array com o caminho é passado para a função *followUntilFindArchive()*, o qual irá procurar pelo arquivo em nossa estrutura de dados. Lembrando que, neste método privado, existe uma verificação para validar se o diretório, ou arquivo, que estamos procurando, existe.

Após encontrado o arquivo desejado, chamamos um método privado *accessAndReadArchive*, que, por sua vez, executa operações para poder ler o conteúdo do arquivo, caso o conteúdo seja vazio, ou não exista, apenas o nome do arquivo é escrito na tela.

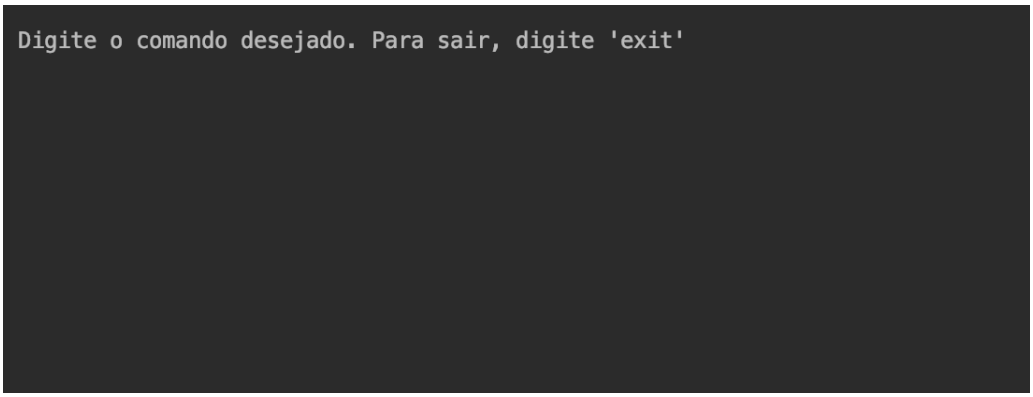
2.7. write

O método write tem como fucionalidade escreer dados em um arquivo criado em nosso sistema de arquivos, para executá-lo precisamos passar por parâmetro 3 dados, sua localização no sistema, qual o conteúdo desejado a ser escrito no arquivo e qual o seu tamanho. Esses dados, então, são passados para a função privada *followUntilWriteArchive*, que procura, recursivamente, pelo arquivo desejado e escreve o conteúdo nele. Caso o arquivo não exista no sistema de arquivos, esta função retorna e avisa o usuário de que o diretório ou arquivo não existe. Dentro deste método privado, quando encontramos o arquivo para escrever, chamamos outro método que executa de fato a escrita no arquivo. Este método se chama *accessAndWriteArchive*.

A função *accessAndWriteArchive* procura pela primeira entrada de diretório vazia para criar o arquivo, caso o diretório esteja cheio um erro é jogado na tela informando que o diretório está cheio caso o contrário encontramos uma entrada de diretorio vazia, e então prossegue com a criação do arquivo. Após encontrada esta entrada procuramos na FAT por uma entrada livre, caso não encontre o método retorna uma mensagem informando que a FAT está cheia. Como o tamanho do dado é passado por parâmetro (variável size), é preciso calcular a quantidade de blocos que será usada para a escrita e então realizar a escrita. Após este fluxo salvamos o estado atual do sistema de arquivos no arquivo *filesystem.dat*.

3. Exemplo de Uso

Para rodar o projeto é necessário ter Java 8 ou maior instalado no computador. Ao rodar o projeto o usuário encontrará a seguinte interface:

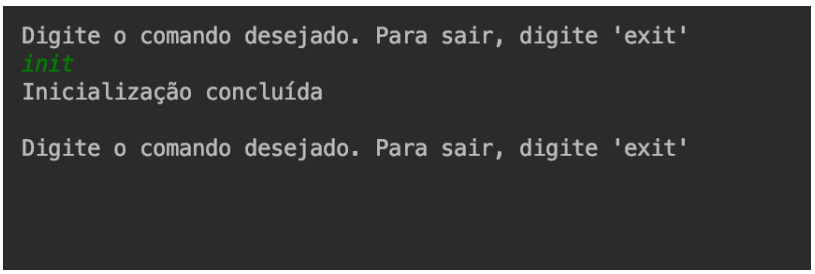


Nesta interface é possível digitar qualquer comando presente na seção de desenvolvimento deste artigo, que são:

- **init**
- **ls**
- **mkdir**
- **create**
- **write**
- **read**

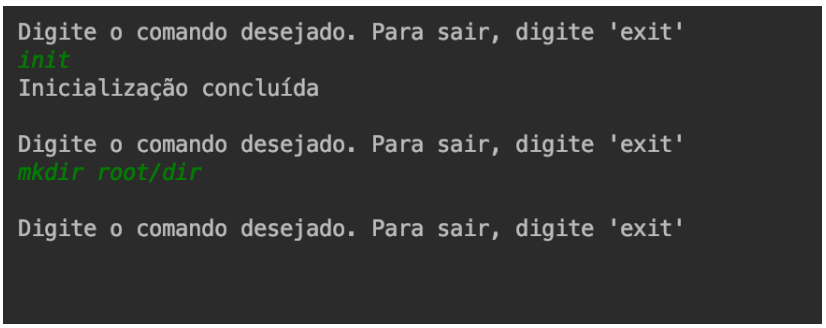
3.1. init

Para executar o comando init basta escrevê-lo no terminal, lembrando que ao executar este comando o sistema de arquivos é zerado.



3.2. mkdir

Para criar um diretório no sistema de arquivos precisamos digitar ”mkdir root/nome_do_diretório”



3.3. create

Para criar um arquivo devemos passar o comando ”create root/dir/nome_do_arquivo content(conteúdo) 200(tamanho)”:

```
Digite o comando desejado. Para sair, digite 'exit'
init
Inicialização concluída

Digite o comando desejado. Para sair, digite 'exit'
mkdir root/dir

Digite o comando desejado. Para sair, digite 'exit'
create root/dir/file.txt Hello 200

Digite o comando desejado. Para sair, digite 'exit'
```

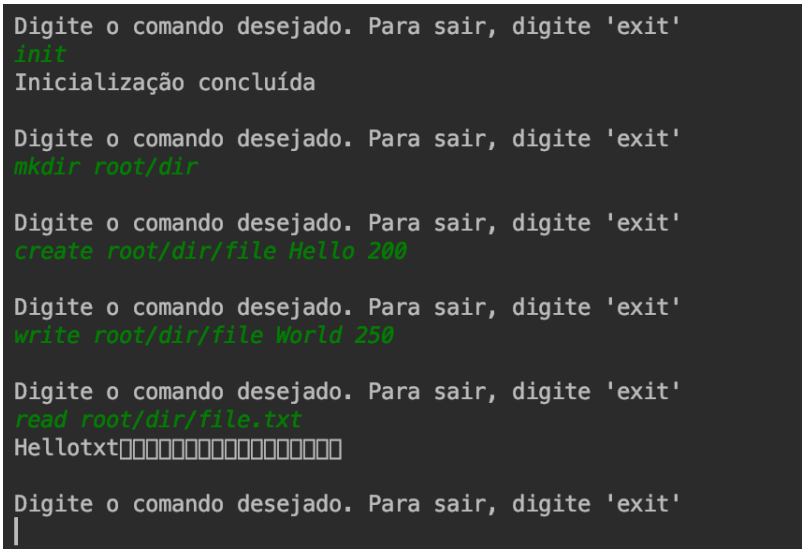
3.4. write

Para escrever em um arquivo já existente devemos passar o comando ”write root/dir/nome_do_arquivo content(conteúdo) 200(tamanho)”:

```
Digite o comando desejado. Para sair, digite 'exit'
init
Inicialização concluída

Digite o comando desejado. Para sair, digite 'exit'
mkdir root/dir

Digite o comando desejado. Para sair, digite 'exit'
create root/dir/file.txt Hello 200

Digite o comando desejado. Para sair, digite 'exit'
write root/dir/file.txt World 250

Digite o comando desejado. Para sair, digite 'exit'
```

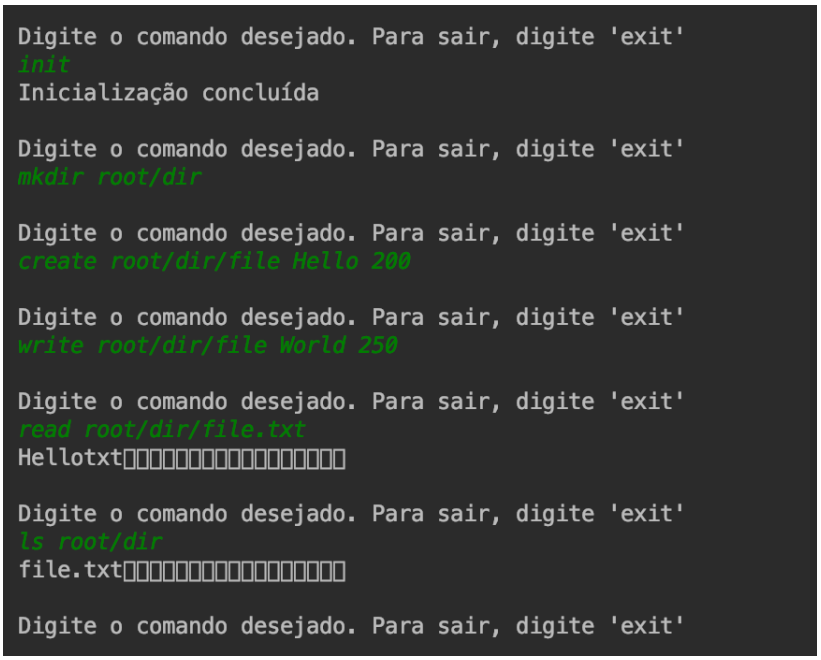
3.5. read

Para ler o conteúdo de um arquivo já existente devemos passar o comando ”read root/caminho/para/diretório”:



3.6. ls

Para listar o conteúdo de um diretório devemos digitar ”ls root/caminho/para/diretório”:



3.7. hexdump

Finalmente para mostrar qual é o conteúdo que se encontra no arquivo *filesystem.dat*, utilizamos a ferramenta hexdump para externalizar o resultado em um terminal, desta forma conseguimos ter uma maneira mais visual sobre como o conteúdo está armazenado em nosso disco.

```
➜  hexdump -C filesystem.dat
00000000  7f fe 7f fe 7f fe 7f fe  7f ff 7f ff 7f ff 7f ff  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001000  64 69 72 00 00 00 00 00  00 00 00 00 00 00 00 00  |dir.............|
00001010  00 00 00 00 00 00 00 00  00 02 00 05 00 00 00 00  |................|
00001020  66 69 6c 65 00 00 00 00  00 00 00 00 00 00 00 00  |file............|
00001030  00 00 00 00 00 00 00 00  00 01 00 07 00 00 00 05  |................|
00001040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001400  66 69 6c 65 2e 74 78 74  00 00 00 00 00 00 00 00  |file.txt........|
00001410  00 00 00 00 00 00 00 00  00 01 00 06 00 00 00 05  |................|
00001420  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001800  48 65 6c 6c 6f 74 78 74  00 00 00 00 00 00 00 00  |Hellotxt........|
00001810  00 00 00 00 00 00 00 00  00 01 00 06 00 00 00 05  |................|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00001c00  57 6f 72 6c 64 00 00 00  00 00 00 00 00 00 00 00  |World...........|
00001c10  00 00 00 00 00 00 00 00  00 02 00 05 00 00 00 00  |................|
00001c20  66 69 6c 65 00 00 00 00  00 00 00 00 00 00 00 00  |file............|
00001c30  00 00 00 00 00 00 00 00  00 01 00 07 00 00 00 05  |................|
00001c40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00200000
```

4. Lições Aprendidas e Problemas Encontrados

4.1. Lições Aprendidas

Durante o decorrer do desenvolvimento do trabalho, tivemos diversas lições aprendidas. Uma delas foi a pouca comunicação entre os integrantes na hora de desenvolver um método do trabalho. Isso fez com que tivessesmos várias visões diferentes que não resolviam o problema que estava em pauta, por exemplo, quando implementamos o método de escrita de arquivos o entendimento do fluxo dos dados nos blocos estava muito superficial e não estávamos conseguindo terminá-lo, porém quando conversamos sobre a implementação em conjunto conseguimos reverter a situação. Agora entendemos que devemos nos comunicar constantemente em prol da tarefa para que ela seja feita com maior qualidade.

4.2. Problemas Encontrados

Como no início do desenvolvimento deste trabalho não tivemos muita comunicação, no decorrer do tempo nos deparamos com situações a qual tivemos alterações em métodos que estavam corretos e passaram a não funcionar mais. Além disso tivemos também uma situação problemática com o arquivo *filesystem.dat*, o qual não estava recebendo as atualizações que fazíamos no sistema. Outro problema que o grupo enfrentou foi que o método ls também estava listando o conteúdo de arquivos, por exemplo, caso executássemos ls root/dir/file.txt o retorno desta função seria o conteúdo do arquivo file.txt. Grande parte de nossos problemas foram resolvidos com a comunicação mais frequente entre os integrantes do time.

5. Conclusão

O objetivo principal desta tarefa é expandir nossos conhecimentos através de um exercício prático e ao finalizar a segunda tarefa prática da cadeira, foi possível observar o aumento de nossos conhecimentos em relação ao conteúdo de sistemas de arquivos. Agora conseguimos compreender melhor como funciona o fluxo de dados quando um arquivo é criado e modificado e qual é o setup inicial do sistema tipo FAT. Através da simulação do sistema de arquivos escrito em Java, pudemos mostrar em forma de código como é a funcionalidade do sistema FAT, apresentando diversos métodos como de criação de diretórios e arquivos e escrita em arquivos. O grupo trabalhou em equipe passando por dificuldades e superando os desafios, fazendo com que fosse possível entregar a tarefa, em geral gostamos muito da tarefa e gostaríamos de ter tido mais tempo para entregá-la com maior qualidade.