BrisaFS Segmentation Fault

Henrique Augusto Santos Batista, Isaias Alves da Silva,
Richard da Cruz Lopes, João Victor Fontinelle Consonni

Maio 2019

1 Introdução

Esse projeto foi implementado para aprofundar os conhecimentos em sistemas operacionais apresentados durante a disciplina de Sistemas operacionais cursada na UFABC o objetivo desse projeto é implementar um sistemas de arquivo a partir de uma base fornecida pelo professor e usando o Filesystem in Userspace (FUSE) que é uma interface em software para sistemas operacionais estilo UNIX que permite que usuários sem direitos de administração criem seus próprios sistemas de arquivos sem a necessidade de editar código do kernel do sistema operacional.

2 Descrição do projeto

Esse sistema implementa os seguintes comandos de controle de arquivos cd, ls, touch, cat, mkdir, rm, rm -r, chown, chgrp e chmod.

Cada operação tem como parâmetro chave o caminho(path) e a partir do path percorre os superblocos a partir da raiz do path e busca subdiretórios/arquivos. Cada arquivo é tratado como uma lista encadeada, vinclulando seu inode ao seu conteúdo, mas todos estão armazenados em um array com acesso direto.

Na estrutura do inode, tem um atributo chamado bloco. Esse atributo contém um número que indica qual é o bloco de dados a qual aquele inode se refere. Quando o valor desse atributo é zero, o bloco é entendido como livre, caso contrário, ele está ocupado.

No caso de arquivos com mais de 4096 bytes, o sistema usa uma estrutura de tabela FAT. Cada inode tem um atributo proxbloco que possui o indice do inode do bloco seguinte. Se este valor for 0, o bloco em questão é o último do arquivo.

3 Modificações Feitas

- Modificações na estrutura do inode, com adição de novos atributos.

Antigo

```
typedef struct {
    char nome[250];
    uint16_t direitos;
    uint16_t tamanho;
    uint16_t bloco;
} inode;
```

Novo

```
typedef struct {
    uint16_t id;
    char nome[220];
    mode_t type;
    uint32_t timestamp[2];
        //0: Modificacao,
        1: Acesso
    uint16_t direitos;
    uint32_t tamanho;
    uint16_t bloco;
    uint16_t proxbloco;
    uid_t userown;
    gid_t groupown;
} inode; // 256 bytes
```

- Modificação em parâmetros definidos no início do código.
- Criadas as funções:
  - armazena_data: Armazenamento de datas e horários de modificação e acesso de um arquivo (milestone 5);
  - salva_disco: salva o disco que está em memória principal para um arquivo em memória secundária (milestone 5);
  - carrega_disco: Carrega um arquivo em memória secundária para a memória principal, se este arquivo existir. Também calcula o espaço livre em disco (milestone 5).
  - quebra_nome: Recebe o path de um arquivo e retorna o nome do arquivo e o path restante (milestone 7).
  - dir_tree: Recebe um caminho indicado por path e retorna o id do inode indicado pelo path (milestone 7).
  - unlink_brisafs: Implementação da remoção de arquivos (milestone 7)
  - chown_brisafs: Alteração de usuário e grupo usando chown e chgrp (milestone 5)
  - chmod_brisafs: Alteração de direitos usando chmod.
  - mkdir_brisafs: Criação de diretórios usando mkdir.
  - rmdir: Remoção de diretórios.
- Modificadas as funções:
  - preenche_bloco: Passou a verificar se existe espaço em disco e se o arquivo que está sendo criado não excede o permitido pelo sistema de arquivos. Também passa a fazer a parte de encontrar um bloco livre para ser preenchido. Passou a considerar a criação de arquivos do tipo diretório e de arquivos maiores do que 4096 bytes. Também atualiza o diretório pai do arquivo que está sendo criado, adicionando o novo arquivo à lista de arquivos do diretório pai (milestone 7).

- init_brisa: Quando não for possível carregar o disco (com a função carrega_disco), passou a criar o diretório raiz (milestone 7), ale de criar um arquivo com informações relativas ao sistema de arquivos, substituindo o arquivo de boas vindas anterior.
- compara_nome: Devolve 1 caso o nome do arquivo seja o mesmo do indicado pelo path e 0 caso contrário (milestone 7).
- getattr_brisafs: Passou a considerar o suporte a diretórios e subdiretórios, além de arquivos e diretórios com o mesmo nome, mas caminhos distintos (milestone 7).
- readdir_brisafs: Passou a considera o suporte a diretórios e subdiretórios, além de arquivos e diretórios com o mesmo nome, mas caminhos distintos (milestone 7).
- write_brisafs: Modificado para suportar a modificação e criação de arquivos que excedam 4096 bytes, além de validar se o tamanho do buffer cabe no arquivo (milestone 7).
- read_brisafs: Modificado para suportar a leitura de arquivos que excedam 4096 bytes.
- truncate_brisafs: Adequação para prever o suporte a subdiretórios e arquivos com mais de 4096 bytes.

3.1 Suporte à criação de diretórios

Para suportar o sistema de diretórios, cada diretório é criado como um arquivo onde podemos fazer a distinção entre arquivos e diretórios através de um campo *tipo* na estrutura do inode. Os inodes referentes aos diretórios também possuem na sua estrutura uma lista que armazena o id de todos os inodes que estão contidos dentro dele na estrutura de diretórios, assim, uma pasta que possua três arquivos e outros dois diretórios, terá na sua lista de diretórios cinco ids de inodes: três apontando para cada um dos arquivos e dois apontando para cada um dos dois diretórios. Dessa forma, somos capazes de listar corretamente o conteúdo de cada diretório sem que hajam erros na estrutura caso existam nomes de diretórios repetidos no disco.

4 Limites do Sistema

O tamanho máximo de arquivo é de até 64MB. Como estamos projetando o nosso sistema de arquivos para suportar, no máximo, 4 arquivos com 64MB, temos 62500 blocos de dados, o que significa 62500 inodes. Desta forma, quando cada arquivo utiliza no máximo um bloco, ou seja, tem 4096 bytes ou menos. O nosso sistema de arquivos consegue suportar até 62500 arquivos.
Quanto ao disco, este possui 62500 blocos de dados, mais 3907 blocos para comportar os inodes, totalizando 66407 blocos. Cada bloco tem 4096 bytes, então o arquivo tem 272003072 bytes, o que resulta em, aproximadamente, 272MB.
O nome do arquivo está limitado pelo atributo nome da estrutura inode. Na nossa implementação, esse atributo é do tipo char e tem 220 bytes, essa escolha foi para garantir que cada inode tivesse exatamente 256 bytes. Assim, cada bloco de 4096 bytes comporta, exatamente, 16 inodes. Desta forma, qualquer

inode do sistema está integralmente contido em um único bloco. Isso não tem nenhum benefício para a implementação como ela está hoje, mas caso o super-bloco de inode estivesse em memória secundária, o acesso seria mais eficiente, pois bastaria ler um bloco para qualquer inode.
Cada diretório possui um bloco de 4096 bytes para comportar o identificador de todos os inodes dentro dele, assim como um contador que indica quantos arquivos estão dentro dele. Como cada identificador, assim como o contador, são do tipo uint16_t, que ocupa 2 bytes, cada diretório suporta 2047 arquivos mais um contador.

5 Como Compilar e Executar

Esse projeto foi implementado usando o FUSE2 então par isso o primeiro passo é instalar as bibliotecas necessárias para o funcionamento a baixo comando para instalação no ubuntu testado na versão 16.04 LTS

```
$ apt-get install gcc make fuse libfuse-dev
```

Após isso clone o repositório do projeto que está no GitHub com o comando a baixo

```
$ git clone https://github.com/ufabc-bcc/2019_Q1_SO_BrisaFS-segmentation-fault.git
```

Então compile o código usando o GCC com o comando a baixo]

```
$ gcc -D_FILE_OFFSET_BITS=64 -Wall -Werror -O3 -g brisafs_v2.c -o brisafs -lfuse
```

Então crie um diretório para servir de base para o seu sistema de arquivos e monte o sistema de arquivos no diretório criado

```
$ mkdir /projetoBrisa
```

```
$ ./brisafs -f -s /projetoBrisa
```

Os parâmetros são os seguintes:

-f Roda em foreground. Interessante durante o desenvolvimento para ver as saídas do programa na stdout

-s Single-threaded, informa ao FUSE que a implementação não aceita chamadas simultâneas. Assim o FUSE garante que só efetuará uma chamada após uma eventual chamada anterior ter sido completada.

/ projetoBrisa Ponto de montagem onde o sistema de arquivos BrisaFS estará acessível

Agora o projeto deve estar rodando em outro terminal acesse a pasta onde foi montado o sistema com o comando a baixo e os comando estarão funcionando.

```
$ cd /projetoBrisa
```

6 Análise de Escalabilidade

O sistema não é escalável. A implementação atual coloca o disco inteiro em memória principal, ou seja, 272MB. Para cada0 criação, deleção e modificação de algum arquivo, todos os 272MB são salvos em um arquivo hdd1, o que garante a persistência do sistema de arquivos. Logo, é evidente que se aumentássemos muito o tamanho do disco, digamos que para um 1GB, cada operação do sistema de arquivos levaria muito mais tempo, além de prejudicar a performance do sistema computacional como um todo.

7 Pontos de Melhoria

O sistema de arquivos poderia melhorar sua escalabilidade ao deixar o disco em memória secundária, deixando apenas o que está sendo utilizado em memória principal. Tal melhoria faria com que o sistema ficasse mais leve e tornaria a implementação dos requisitos do milestone 10, muito mais fáceis.

8 Conclusão

Ao finalizar a implementação desse projeto pudemos alcançar os milestones 5 e 7 definidos pelo professor e como citado na seção 7 faltou as operações funcionarem acessando a memória secundaria para possibilitar a realização da milestone10 completando o projeto porem durante a implementação, realizamos a implementação da função chmod que altera as permissões de um arquivo para alguma diferente da padrão e da função rm -r que possibilita a remoção de diretórios.
A implementação desse projeto permitiu ao grupo como um todo desenvolver habilidades em desenvolvimento e aprofundar o conhecimento obtido durante a disciplina