Sistemas Operacionais – 2024
Terceiro Trabalho
Unix File System

1 Requisitos Gerais

Neste projeto, você implementará um sistema de arquivos simples, semelhante ao UNIX, com uma estrutura de diretórios hierárquica. Os arquivos aumentam e os diretórios aumentam e diminuem, portanto, você precisa gerenciar o espaço livre em disco. Você poderá navegar na estrutura de diretórios, criar novos arquivos e diretórios, excluí-los, etc. As funcionalidades são semelhantes aos sistemas de arquivos UNIX, mas não incluem permissão e gerenciamento de usuários. Também não exigimos simultaneidade ou alto desempenho do seu sistema de arquivos. Em outras palavras, você pode assumir que ele é acessado apenas por um processo por vez.

2 Recursos

O livro texto da disciplina possui informações que o ajudarão a concluir este projeto, mas também recomendamos a seção *The File System* no artigo de Ritchie & Thompson sobre o sistema de compartilhamento de tempo UNIX de 1974, disponível no AVA e em:
`http://cacm.acm.org/magazines/1974/7/11695-the-unix-time-sharing-system/abstract`.

3 API

Nós fornecemos um código inicial, incluindo várias funções para acessar o disco (consulte `block.h` para obter detalhes) e também vários comandos do interpretador de comandos (explicados posteriormente) para testar o sistema de arquivos. Você deve implementar a maioria das chamadas de sistema de arquivos UNIX, conforme listadas a seguir:

3.1 Inicialização do módulo do sistema de arquivos

Protótipo: `void fs_init(void);`
Descrição da função: Essa função inicializa as estruturas de dados e os recursos usados pelo subsistema do sistema de arquivos e, se detectar que o disco já está formatado, faz sua montagem automaticamente no diretório raiz. Ela é chamada no momento da inicialização do sistema, mas antes do módulo de bloco (`block.c`) ser inicializado. Então, você precisa chamar `block_init` em `fs_init`. Observe que no momento em que `fs_init` é chamada, o disco não está necessariamente formatado. Como resultado, você precisa criar um mecanismo para que um disco formatado seja reconhecido (consulte `fs_mkfs`).

3.2 Formatação de disco

Protótipo: `int fs_mkfs(void);`
Descrição da função: Esta função formata um disco: um disco bruto ou um que foi formatado anteriormente. Em seguida, ela monta o sistema de arquivos recém-formatado no diretório raiz. O tamanho do disco (em blocos) é definido como `FS_SIZE` em `fs.h`. (Você pode aumentar o tamanho do disco para teste, se desejar.) Assumimos que existe um e apenas um disco presente no sistema. Você precisa escrever um número mágico no super bloco para que mais tarde possa ser reconhecido como um disco formatado (ou

seja, você ainda pode acessar um disco formatado e obter seu conteúdo após o interpretador de comandos terminar e reiniciar). A função deve retornar 0 em caso de sucesso e -1 em caso de falha.

3.3 Abertura e, possivel, criação de arquivo

Protótipo: `int fs_open(char *filename, int flags);`

Descrição da função: Dado um nome de arquivo (*filename*), `fs_open()` retorna um descritor de arquivo, um número inteiro pequeno e não negativo para uso em chamadas de sistema subsequentes (`fs_read`, `fs_write`, `fs_lseek`, etc.). O descritor de arquivo retornado por uma chamada bem-sucedida será o descritor de arquivo com o número mais baixo que não está aberto no momento.

O parâmetro *flags* deve incluir um dos seguintes modos de acesso: `FS_O_RDONLY`, `FS_O_WRONLY`, `FS_O_RDWR`. Eles solicitam a abertura do arquivo somente leitura, somente gravação ou leitura/gravação, respectivamente. As constantes são definidas no arquivo `common.h`.

Abrir um arquivo retorna um novo descritor de arquivo ou -1 se ocorreu um erro.

Se um arquivo inexistente for aberto para gravação, ele deverá ser criado. Uma tentativa de abrir um arquivo inexistente somente para leitura deve falhar.

Para facilitar sua vida, assumimos que o nome do arquivo (*filename*) passado para as chamadas de sistema pode ser apenas ".", "..", um diretório ou um nome de arquivo no diretório atual. Portanto, você não precisa analisar o caminho com nomes de diretório e arquivo separados por "/". Você também pode supor que o comprimento do nome do arquivo (e do nome do diretório) seja menor que 32 bytes (`MAX_FILE_NAME`). Essas suposições permanecem as mesmas para as funções seguintes.

É considerado um erro abrir um diretório em qualquer modo além do `FS_O_RDONLY`.

Você pode usar uma tabela de descritor de arquivo compartilhada. Você não precisa se preocupar com gerenciamento de usuários ou listas de controle de acesso.

3.4 Fechamento de descritor de arquivo

Protótipo: `int fs_close(int fd);`

Descrição da função: `fs_close()` fecha um descritor de arquivo, para que ele não se refira mais a nenhum arquivo e possa ser reutilizado. Retorna zero em caso de sucesso e -1 em caso de falha. Se o descritor foi a última referência a um arquivo que foi removido usando a desvinculação (**unlink**), o arquivo será excluído.

3.5 Leitura em arquivo

Protótipo: `int fs_read(int fd, char *buf, int count);`

Descrição da função: `fs_read()` tenta ler até (*count*) bytes do descritor de arquivo *fd* no buffer, indicado por *buf*. Se *count* for zero, `fs_read()` retornará zero e não executa nenhuma outra ação. Em caso de sucesso, o número de bytes lidos com êxito é retornado e a posição do arquivo é avançada por esse número. Não é um erro se esse número for menor que o número de bytes solicitados; isso pode acontecer, por exemplo, porque menos bytes estão disponíveis no momento. Em caso de erro, -1 é retornado. No caso de erro, não é necessário alterar ponteiro de deslocamento do arquivo.

3.6 Escrita em arquivo

Protótipo: `int fs_write(int fd, char *buf, int count);`

Descrição da função: `fs_write()` grava *count* bytes no arquivo referenciado pelo descritor de arquivo *fd* do buffer indicado por *buf*.

Em caso de sucesso, o número de bytes gravados de *buf* é retornado (um número menor que *count* pode ser retornado) e a posição do arquivo é avançada por esse número. Em caso de erro, -1 é retornado. É um erro tentar gravar em um ponto além do tamanho máximo de um arquivo. É um erro se a contagem for maior que zero, mas nenhum byte for gravado.

Um arquivo de tamanho zero não deve ocupar nenhum bloco de dados.

Se *count* for zero, 0 será retornado sem causar outros efeitos.

3.7 Reposicionamento do deslocamento para leitura/gravação

Protótipo: `int fs_lseek(int fd, int offset);`

Descrição da função: A função `fs_lseek()` reposiciona o deslocamento do arquivo aberto associado ao descritor de arquivo *fd* para o deslocamento (*offset*) do argumento.

A função `fs_lseek()` permite que o deslocamento do arquivo seja definido além do final do arquivo (mas isso não altera o tamanho do arquivo). Se os dados forem gravados posteriormente nesse deslocamento, o arquivo será preenchido com '\0' no espaço intermediário.

Após a conclusão bem-sucedida, `fs_lseek()` retorna o local de deslocamento resultante conforme medido em bytes desde o início do arquivo. Caso contrário, o valor -1 será retornado.

3.8 Criação de diretório

Protótipo: `int fs_mkdir(char *dirname);`

Descrição da função: `fs_mkdir()` tenta criar um diretório chamado *dirname*. Retorna zero em caso de sucesso ou -1 se ocorreu um erro. `fs_mkdir()` deve falhar se o diretório *dirname* já existir.

Novos diretórios devem conter entradas "." e "..". É um erro tentar criar um diretório sem eles.

3.9 Remoção de diretório

Protótipo: `int fs_rmdir(char *dirname);`

Descrição da função: `fs_rmdir()` exclui um diretório que deve estar vazio. Em caso de sucesso, zero é retornado; em caso de erro, -1 é retornado (por exemplo, tentando excluir um diretório não vazio).

3.10 Mudança do diretório atual

Protótipo: `int fs_cd(char *dirname);`

Descrição da função: `fs_cd()` altera o diretório atual para o especificado em *dirname*. Em caso de sucesso, zero é retornado. Em caso de erro, -1 é retornado.

3.11 Criação de um novo nome para um arquivo

Protótipo: `int fs_link(char *oldpath, char *newpath);`

Descrição da função: `fs_link()` cria um novo link (também conhecido como link físico) para um arquivo existente com o nome *oldpath*. Se já existir um *newpath*, ele não será substituído. O novo nome pode ser usado exatamente como o antigo para qualquer operação; os dois nomes se referem ao mesmo arquivo e é impossível dizer qual nome era o "original".

Em caso de sucesso, zero é retornado. Em caso de erro, -1 é retornado. É um erro usar esta função em um diretório.

Observe que, como não existem "caminhos" além do diretório atual, o pai ou o diretório filho, *oldpath* e *newpath* são na verdade ambos nomes de arquivos e só podem estar no mesmo diretório.

3.12 Remoção de arquivo

Protótipo: `int fs_unlink(char *filename);`

Descrição da função: `fs_unlink()` exclui um nome do sistema de arquivos. Se esse nome foi o último link para um arquivo e nenhum processo o abriu, o arquivo será excluído e o espaço usado estará disponível para reutilização.

Se o nome for o último link para um arquivo, mas ele ainda estiver aberto em um descritor de arquivo existente, ele permanecerá até que o último descritor de arquivo referente a ele seja fechado.

Em caso de sucesso, zero é retornado. Em caso de erro, -1 é retornado. É um erro usar esta função em um diretório.

3.13 Status de arquivo/diretório

Protótipo: `int fs_stat(char *filename, fileStat *buf);`

Descrição da função: `fs_stat()` retorna informações sobre um arquivo. Ela retorna uma estrutura **fileStat** (definida em `common.h`), que contém os seguintes campos:

```
typedef struct{
     int inodeNo; /* número do i-node
do arquivo */
     short type; /* tipo do i-node do
arquivo: DIRECTORY, FILE_TYPE (há outro
valor FREE_INODE que nunca aparece aqui */
     char links; /* número de links para
o i-node */
     int size; /* tamanho do arquivo em
bytes */
     int numBlocks; /* número de blocos
usados pelo arquivo */
}fileStat;
```

Não reutilize essa estrutura para seus inodes.

Nota: se sua implementação precisar de membros diferentes, você poderá modificar essa estrutura, mas adicione-a apenas e atualize os membros que já estão lá.

3.14 Listar o conteúdo atual do diretório

Protótipo: `void shell_ls(void);`

Descrição da função: Observe que esta função está em `shell.c`.

`shell_ls()` imprime uma lista de arquivos no diretório atual. A lista não precisa ser classificada em ordem alfabética.

4 Comandos da shell

Você poderá testar a implementação do sistema de arquivos por meio dos comandos da shell (fornecidos em `shell.c`) apresentados na Tabela 1. Também testaremos seu sistema de arquivos por meio deste interpretador de comandos; portanto, não altere os comandos existentes. Você pode adicionar novos comandos na shell para seus próprios fins de teste.

5 Fragmentação de Diretório

Assim que as entradas de um diretório couberem em um número menor de blocos que o alocado para o diretório, você deve fazer com elas caibam. Nesse caso, um bloco de dados deve ser liberado e os buracos preenchidos.

Por exemplo, digamos que uma implementação possa acomodar quatro entradas em um bloco de dados. Atualmente, existem onze entradas em algum diretório, o que requer três blocos de dados. Quando quaisquer três entradas forem removidas do diretório, os dados do diretório devem ser consolidados para caber em dois blocos de dados.

6 Teste

Você pode testar seu código em Linux digitando **make** e executando **./lnxsh**. Este programa irá ler/gravar em um arquivo chamado **disk**.

Será fornecido um script de teste que você poderá usar para testar seu sistema de arquivos. Você também escreverá seus próprios casos de teste e os enviará com seu código.

Tabela 1: Comandos da Shell

Comando	**Argumentos**	**Descrição**
mkfs		Cria um novo sistema de arquivos, ou seja, formata o disco para que ele fique pronto para outras operações do sistema de arquivos.
open	<filename> <flag>	Abre um arquivo com as <flag> fornecidas, retorna um descritor de arquivo (fd) associado a esse arquivo. <flags>: 1: `FS_O_RDONLY`; 2: `FS_O_WRONLY`; 3: `FS_O_RDWR`. A flag é a mesma flag usada pela chamada de sistema quando um arquivo é aberto. O deslocamento do arquivo atual será 0 quando o arquivo for aberto.
read	<fd> <size>	Lê <size> bytes do arquivo associado a <fd>, a partir do deslocamento atual do arquivo. O deslocamento atual do arquivo avançará <size> bytes após a leitura.
write	<fd> <string>	Escreve <string> no arquivo associado a <fd>, a partir do deslocamento atual do arquivo. O deslocamento atual do arquivo avançará <size> bytes após a gravação.
lseek	<fd> <offset>	Move o ponteiro de deslocamento do arquivo atual associado a <fd> para um novo deslocamento de arquivo em <offset>. O <offset> significa o número de bytes desde o início do arquivo.
close	<fd>	Fecha o arquivo associado ao <fd>.
mkdir	<dirname>	Cria um subdiretório <dirname> no diretório atual.
rmdir	<dirname>	Remove o subdiretório <dirname>.
cd	<dirname>	Altera o diretório atual para <dirname>.
link	<src> <dest>	Cria um link chamado <dest> para um arquivo existente chamado <src>.
unlink	<name>	Remove um link para <name>. (Quando a contagem de links cair para 0, exclua o arquivo ou diretório).
stat	<name>	Mostra o status do arquivo ou diretório com o nome <name>. Deve exibir suas informações de inode; se é um arquivo ou diretório; sua contagem de links; tamanho do arquivo/diretório; número de blocos alocados; outras informações armazenadas sobre este arquivo/diretório.
ls		Mostra o conteúdo do diretório atual. **Nota!** Você deve implementar esta função.
cat	<filename>	Mostra o conteúdo do arquivo.
create	<filename> <size>	Cria um arquivo chamado <filename> dentro do diretório atual e preencha-o com <size> de dados e feche o arquivo. (Para fins de teste)

Como a saída da sua implementação deve ser semelhante à saída dos testes fornecidos, o envio final não deve gerar mensagens de erro personalizadas.

7 Arquivos

Você deve usar o modelo de código como ponto de partida para o seu projeto. Você só precisa alterar três arquivos: `fs.h`, `fs.c` e `shell.c`. No entanto, fique à vontade para adicionar outros arquivos para que você possa organizar melhor seu código ou modificar os arquivos existentes, mas certifique-se de descrever no seu relatório quaisquer arquivos adicionados ou alterações feitas em outros arquivos.

Você deve usar as funções fornecidas em `block.h` para ler e gravar blocos no sistema de arquivos. Essas funções usarão um arquivo em Linux.

Os seguintes arquivos são compilados e ligados para gerar um executável em Linux:

- `shellutilFake.c`: Fake para algumas funções auxiliares do interpretador de comandos;

- `blockFake.c`: usa um arquivo UNIX como um disco;

8 Avaliação

Seu projeto será avaliado com base nos seguintes itens:

- Casos de teste fornecidos;
- Descrição do projeto do seu sistema de arquivos;
- Seus casos de teste;
- Casos de teste fornecidos;

Os seus casos de teste enviados devem estar da mesma forma que os testes fornecidos: um script Python. Indique a finalidade de cada teste (como um comentário no arquivo é suficiente). Seus testes não precisam ser tão extensos quanto os casos fornecidos. Esta é sua oportunidade de testar coisas que os casos fornecidos não analisam e que podem ser específicas para sua implementação (ou seja, não definidas nas especificações). Se seus testes forem executados de maneira diferente da dos testes fornecidos, indique isso no seu documento de design.

Além da correção do programa, o professor e/ou assistente de ensino poderão fazer perguntas durante uma apresentação do trabalho. Durante a apresentação, o grupo deverá explicar o funcionamento do programa e responder a perguntas relativas ao projeto.

9 Entrega do Trabalho

O trabalho pode ser feito em **grupo de no máximo três alunos** e deve ser entregue até o dia **5 de julho de 2024**. Não haverá prorrogação devido ao final do semestre. Portanto, programe suas atividades de acordo para entregar o trabalho até essa data.

Além do código fonte devidamente comentado, o grupo deve enviar um relatório que deve incluir detalhes sobre o projeto do sistema de arquivos. O documento deve fornecer uma descrição abrangente e concisa do seu projeto e deve conter pelo menos as seguintes seções:

- Visão geral do sistema de arquivos;
- Super bloco;
- Inodes;
- Estrutura e gerenciamento dos blocos de dados livres;
- Implementação de link/unlink (como você fez isso?);

O relatório deve estar em formato PDF. Se o que você está descrevendo é melhor explicado com um diagrama, crie um (ou mais). Limite o seu documento a cinco páginas.

Tanto o relatório quanto os arquivos fontes da implementação devem ser colocados em um arquivo ZIP e enviados via AVA. Arquivos em outros formatos, como RAR, não serão considerados e o trabalho não será avaliado. Inclua em seu arquivo ZIP apenas os arquivos fontes e Makefile, ou seja, não inclua arquivos compilados (.o).