34.110-0 – Laboratório de Sistemas Operacionais

Turma A

2010/2

Projeto Final: Sistemas de Arquivos

1 Objetivo

Entender mais a fundo a construção de software básico, construindo um sistema de arquivos elementar a partir de um dispositivo de disco genérico.

2 Visão Geral

Um sistema de arquivos transforma um dispositivo de disco que armazena setores de dados em uma abstração de mais alto nível: arquivos. Um dispositivo de disco permite apenas guardar e recuperar setores com um número fixo de bytes. Cabe ao programador do sistema de arquivos organizar estes setores, de forma a dividi-los entre todos os dados a serem armazenados, alocá-los ou desalocá-los a medida que os dados crescem ou diminuem, gerenciar o espaço livre, etc. Arquivos, por outro lado contém uma sequencia contínua de bytes com tamanho variável. O programador de aplicação não se preocupa com como estes dados são guardados, como o arquivo cresce ou diminui, em quais setores estão os dados, etc. Arquivos são localizados por nome, e a sua enumeração, criação e remoção são tarefas bem simples.

Neste projeto nós iremos construir um sistema de arquivos sobre um dispositivo de disco simulado, para prover os serviços necessários ao funcionamento de um interpretador de comandos elementar. Uma visão esquemática é a seguinte:

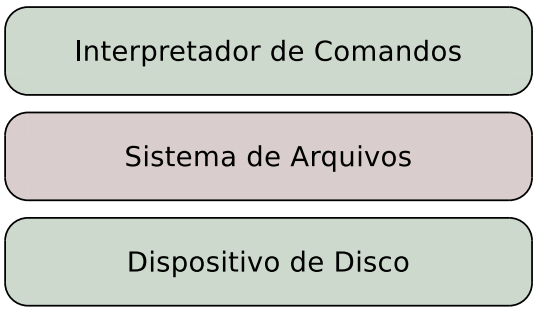

Figura 1: Visão Geral da Arquitetura de Software

Neste diagrama, cada camada é implementada utilizando apenas as funções fornecidas pela camada inferior. A implementação do dispositivo de disco e do interpretador de comandos, além da especificação do sistema de arquivos, serão fornecidos pelo professor.

3 Tarefas

Neste projeto a sua tarefa é implementar as funções usadas pelo interpretador de comandos usando como base as funções fornecidas pelo dispositivo de disco. O código fonte fornecido já está estruturado com implementações vazias destas funções, falta completar a sua implementação. O sistema de arquivos implementado deve seguir a especificação contida neste documento, em especial as simplificações e limitações dadas para tornar a tarefa mais simples.

As atividades a serem cumpridas são:

1. Estudar a especificação do sistema de arquivos a ser implementado, dada neste documento.
2. Baixar o código fornecido pelo professor e estudar a sua organização, procurando localizar onde devem ser implementadas as funções pedidas.
3. Implementar as funções do sistemas de arquivos, usando os comandos do interpretador de comandos como testes. As funções se separam em dois grandes grupos, nesta ordem:
   (a) Funções de gerência de arquivos;
   (b) Funções de escrita e leitura de arquivos.

   Procure implementar as funções nesta ordem.

4 Entregas

Você deve entregar:

- O código do seu sistema de arquivos, contido no arquivo `fs.c` . Mudanças em outros arquivos estão **proibidas**, logo entregue apenas o arquivo pedido.
- Um relatório descrevendo como as funcionalidades foram implementadas. Esta descrição deve evitar reproduzir as operações que o código realiza, para isto o professor pode ler o mesmo. As descrições devem dar uma visão mais ampla de como as funcionalidades foram implementadas, as estruturas de dados usadas e os princípios de funcionamento da implementação. Procure organizar bem o seu relatório. Uma sugestão é concentrar as descrições por grupo de funções (gerência de arquivos e escrita e leitura de arquivos).

Este projeto deve ser realizado em grupo. Os grupos devem informar ao professor, por e-mail, a composição dos grupos, **impreterivelmente** até o dia 12/11.

As entregas devem ser feitas via Moodle, em um único arquivo .zip ou .tar.gz. O relatório deve estar em formato PDF.

5 Especificação do Ambiente

Antes de descrever a especificação do sistema de arquivos em si, vamos descrever a camada que usará o sistema de arquivos (interpretador de comandos) e a camada que fornecerá o ambiente básico para o o sistema de arquivos (dispositivo de disco).

5.1 Interpretador de Comandos

O interpretador de comandos permite interagir com a implementação do sistemas de arquivos. Esta camada "combina" várias camadas de um SO real (interface de chamada de sistemas, bibliotecas, interpretador de comandos, etc.) em um único componente simplificado. A implementação do interpretador está no arquivo `shell.c`, que deve ser consultado para ver como as funções do sistema de arquivos são usadas.

Também é possível usar os comandos do interpretador para testar a sua implementação. O interpretador de comandos implementa as seguintes comandos, digitados no teclado:

format: Formata o sistema de arquivos.

list: Lista o conteúdo do sistema de arquivos.

create FILE: Cria o arquivo FILE com tamanho 0. Útil para testar o diretório.

remove FILE: Remove o arquivo FILE.

copy FILE1 FILE2: Copia o arquivo FILE1 para o arquivo FILE2.

copyf REAL_FILE FILE: Copia um arquivo "real" chamado REAL_FILE para dentro do ambiente simulado com nome FILE.

copyt FILE REAL_FILE: Copia um arquivo FILE no ambiente simulado para o um arquivo "real" com nome REAL_FILE.

5.2 Dispositivo de Disco

O dispositivo de disco representa um disco rígido idealizado. Ele permite que o programador leia e escreva setores de dados de tamanho fixo (512B). Os setores são numerados, começando em zero e indo até o último setor do disco.

O disco é implementado usando-se um arquivo de imagem armazenado no sistema. O tamanho deste arquivo define o tamanho do disco simulado. O código fornecido já contém funções para criação de arquivo de imagem com um tamanho apropriado. O código desta camada pode ser encontrada no arquivo `disk.c`.

A camada do dispositivo de disco fornece as seguintes constantes e funções ao programador do sistema de arquivos. Todas estas funções, quando não explicitamente especificado, retornam 1 quando a operação completou com sucesso e 0 em caso de erro. Qualquer erro encontrado é impresso na tela, logo os clientes destas funções precisam apenas detectar os erros e não relatá-los.

`SECTORSIZE`**:** Constante que dá o tamanho do setor.

`int bl_init(char *image, int size)`**:** Inicia o dispositivo de disco para acessar a imagem contida no arquivo `image`. Se a imagem não existe, ela é criada com o tamanho dado por `size` setores. Se a imagem já existe, ela é aberta e o argumento `size` é ignorado. Esta é uma função interna do dispositivo de disco, chamada automaticamente pelo interpretador de comandos. O código do sistema de arquivos *não* precisa chamar esta função.

`int bl_size()`**:** Retorna o tamanho do dispositivo de disco *em setores*. O tamanho em bytes é este tamanho multiplicado por `SECTORSIZE`.

`int bl_write(int sector, char *buffer)`**:** Escreve no setor de número `sector` as primeiras `SECTORSIZE` posições do vetor `buffer`, que deve ter pelo menos este número de elementos.

`int bl_read(int sector, char *buffer)`: Escreve nas primeiras `SECTORSIZE` posições do vetor `buffer`, que deve ter pelo menos este número de elementos já alocados, o setor de número `sector` .

6 Especificação do Sistema de Arquivos

As especificação do sistema de arquivos tem duas partes: a descrição de sua interface de programação e a descrição de sua estrutura interna. A interface de programação são as funções, usadas pelo interpretador de comandos, que devem ser implementadas. A estrutura interna define restrições em como estas funções devem ser implementadas, na forma da definição de um sistema de arquivos específico, semelhante à FAT.

6.1 Funções

Implementar o sistema de arquivos corresponde a implementar corretamente as funções de sua interface, descritas a seguir. Nenhuma destas funções seleciona o dispositivo de disco, podemos supor que apenas um dispositivo de disco pode ser usado de cada vez e que quando o sistema de arquivos inicia este dispositivo já está operacional.

Todas estas funções, quando não explicitamente especificado, retornam 1 quando a operação completou com sucesso e 0 em caso de erro. Qualquer erro encontrado deve ser impresso na tela, para permitir que os clientes destas funções possam apenas detectar os erros sem relatá-los. O seu sistema de arquivos deve verificar, para todas as funções, se o disco está formatado e retornar erros de acordo.

6.1.1 Funções de Gerência de Arquivos

`int fs_init()`: Inicia o sistema de arquivos e suas estruturas internas. Esta função é automaticamente chamada pelo interpretador de comandos no início do sistema. Esta função deve carregar dados do disco para restaurar um sistema já em uso e é um bom momento para verificar se o disco está formatado.

`int fs_format()`: Inicia o dispositivo de disco para uso, iniciando e escrevendo as estruturas de dados necessárias.

`int fs_free()`: Retorna o espaço livre no dispositivo em bytes.

`int fs_list(char *buffer, int size)`: Lista os arquivos do diretório, colocando a saída formatada em `buffer`. O formato é simples, um arquivo por linha, seguido de seu tamanho e separado por dois tabs. Observe que a sua função não deve escrever na tela.

`int fs_create(char *file_name)`: Cria um novo arquivo com nome `file_name` e tamanho 0. Um erro deve ser gerado se o arquivo já existe.

`int fs_remove(char *file_name)`: Remove o arquivo com nome `file_name`. Um erro deve ser gerado se o arquivo não existe.

6.1.2 Funções de Escrita e Leitura de Arquivos

`int fs_open(char *file_name, int mode)`: Abre o arquivo `file_name` para leitura (se `mode` for `FS_R`) ou escrita (se `mode` for `FS_W`). Ao abrir um arquivo para leitura, um erro deve ser gerado se o arquivo não existe. Ao abrir um arquivo para escrita, o arquivo deve ser

criado ou um arquivo pré-existente deve ser apagado e criado novamente com tamanho 0. Retorna o identificador do arquivo aberto, um inteiro, ou -1 em caso de erro.

`int fs_close(int file):` Fecha o arquivo dado pelo identificador de arquivo `file`. Um erro deve ser gerado se não existe arquivo aberto com este identificador.

`int fs_write(char *buffer, int size, int file):` Escreve `size` bytes do `buffer` no arquivo aberto com identificador `file`. Retorna quantidade de bytes escritos (0 se não escreveu nada, -1 em caso de erro). Um erro deve ser gerado se não existe arquivo aberto com este identificador ou caso o arquivo tenha sido aberto para leitura.

`int fs_read(char *buffer, int size, int file):` Lê no máximo `size` bytes no `buffer` do arquivo aberto com identificador `file`. Retorna quantidade de bytes efetivamente lidos (0 se não leu nada, o que indica que o arquivo terminou, -1 em caso de erro). Um erro deve ser gerado se não existe arquivo aberto com este identificador ou caso o arquivo tenha sido aberto para escrita.

6.2 Organização

A implementação do sistemas de arquivos *deve* seguir a estrutura descrita a seguir. Resumidamente, é um sistema que usa tabela de alocação de arquivos (FAT) em memória, com agrupamentos (blocos) de 4096 bytes (8 setores), contador de agrupamentos de 16 bits e tamanho de arquivo de 32 bits (um número de 16 bits é um `short` e um de 32 bits é um `int` na maioria das arquiteturas). A FAT tem tamanho fixo, igual ao máximo tamanho de disco suportado com os parâmetros acima (256MiB), e armazena os encadeamentos dos arquivos e a lista de agrupamentos livres. Os arquivos estão organizados em um único diretório, que comporta no máximo 128 arquivos.

O sistema de arquivos nos setores do disco tem a seguinte estrutura:

Figura 2: Leiaute do Disco

O sistema de arquivos começa com a tabela de alocação de arquivos. Esta tabela localiza os agrupamentos utilizados pelos arquivos e corresponde a um vetor de tamanho fixo com 2^{16} posições. Como iremos trabalhar com agrupamentos de 4096 bytes, este vetor ocupa 2^{16} * 2 bytes = 128kiB / 4kiB = 32 agrupamentos (256 setores), de forma contígua logo no começo do disco.

Cada posição nesta tabela de alocação de arquivos corresponde a um agrupamento no disco. Estes 2^{16} agrupamentos endereçam, no máximo, 2^{16} x 4kiB = 256MiB, este sendo o tamanho do maior disco suportado. O *conteúdo* de uma posição na tabela é o índice do próximo agrupamento em um arquivo. Alguns agrupamentos são marcados com indicadores especiais (este indicadores usam números de agrupamento ocupados pela FAT, logo não podem ser usados em arquivos normais):

Valor	Significado
1	Agrupamento Livre
2	Último Agrupamento de Arquivo
3	Agrupamento da FAT
4	Agrupamento do Diretório

Além de estar armazenada no disco, esta tabela deve estar sempre em memória. Em memória, a FAT pode ser declarada assim:

```
unsigned short fat[65536];
```

O sistema de arquivos possui apenas um nível de diretórios. Os arquivos tem nomes alfanuméricos (iguais às regras do sistema de arquivos do Linux) de no máximo 24 caracteres (não se preocupe com validação de nomes, exceto o tamanho). O diretório é um único agrupamento (4kiB), que fica no disco logo após a FAT. Este agrupamento pode ser visto como um vetor de 128 posições da seguinte estrutura de 32 bytes:

```
typedef struct {
       char used;
       char name[25];
       unsigned short first_block;
       int size;
} dir_entry;
```

`used`: Um booleano que indica se a entrada está em uso.

`nome`: O nome do arquivo.

`first_block`: O primeiro agrupamento do arquivo.

`size`: O tamanho do arquivo em bytes.

Como a tabela de alocação, o diretório deve existir no disco e na memória. Na memória o diretório pode ser declarado assim:

```
dir_entry dir[128];
```

A próxima figura mostra um exemplo de uma FAT e o seu disco correspondente. Nesta FAT temos dois arquivos apenas. O primeiro arquivo ocupa o primeiro agrupamento passível de ser ocupado (33), ou seja, aquele que vem logo após a FAT e o diretório no disco. Este arquivo ocupa apenas um agrupamento, logo o mesmo contém o valor 2 na FAT na posição 33. No disco devem estar os dados deste arquivo. O segundo arquivo ocupa dois agrupamentos (34 e 36). Na FAT o agrupamento 34 contém o índice do próximo agrupamento (36), que por sua vez contém o valor 2 indicando que é o último agrupamento do arquivo.

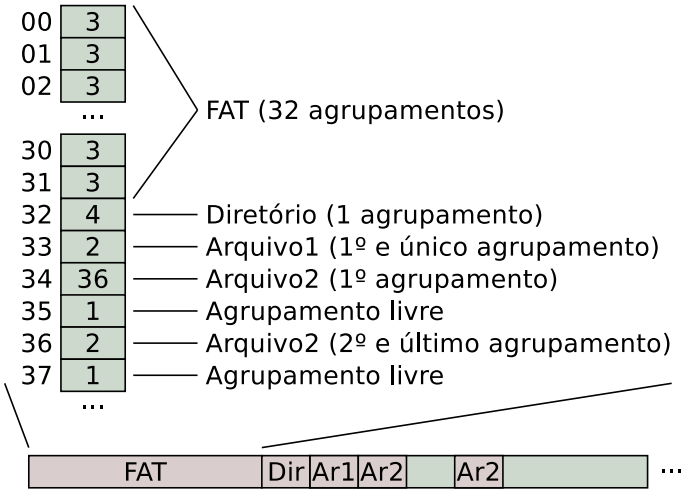

Figura 3: Exemplo de FAT e Disco

7 Dicas

- Não confunda a estrutura da FAT e a estrutura do disco. A FAT reflete a estrutura do disco, indicando os agrupamentos em uso e o encadeamento dos arquivos, porém os dados vão nos setores reais do disco.
- Para acessar o primeiro setor referente ao agrupamento de número x, basta fazer x * 8.
- Não se esqueça que, apesar de mantidos em memória, a FAT e o diretório devem ser escritos em disco a cada modificação, sob o risco de deixar o seu sistema de arquivos inconsistente.
- Podemos tratar qualquer estrutura de dados como um vetor de caracteres, basta fazer um *cast* de ponteiros e usar o tamanho correto. Por exemplo:

```
short fat[65536];
char *buffer;
int i;

buffer = (char *) fat;
for (i = 0; i < sizeof(fat); i++) {
    printf("%c\n", buffer[i]);
}
```

- Podemos pegar qualquer fração de um vetor ajustando o ponteiro e o tamanho desejado. Por exemplo, para passar como argumento para uma função, que espera como argumentos um vetor e um tamanho, os elementos de 5 a 7 (inclusive) de um vetor:

```
char array[256];

function(&array[4], 3);
```

- Para verificar se o arquivo de imagem de disco está sendo escrito de forma correta, use um editor hexadecimal (também chamado de edito binário). Use o GHex ou escolha um em `http://en.wikipedia.org/wiki/Comparison_of_hex_editors`.