UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA

INF01142 - Sistemas Operacionais I N - 2016/2

Trabalho Prático 2

T2FS - Task 2 File System

Prof. Dr. Alexandre da Silva Carissimi

Gustavo Madeira Santana - 252853
Cristiano Salla Lunardi - 240508

1 Questionário

Indique, para CADA UMA das funções que formam a biblioteca libt2fs, se as mesmas estão funcionando corretamente ou não.

```
int identify2 (char *name, int size);
```

Implementado e funcionando corretamente.

```
FILE2 create2 (char *filename);
```

Implementado e funcionando corretamente. Procura pelo caminho indicado pelo filename. Se os diretórios no caminho já existem, é criado um arquivo ali, alocando um record com os dados correspondentes e atualizando os bitmaps de blocos e inodes.

```
int delete2 (char *filename);
```

Implementado e funcionando. Procura pelo arquivo indicado pelo filename. Se existe, seus inode/-record são marcados como inválidos, e os bitmaps de blocos e inodes tem os bits referente ao arquivo marcados como livres.

```
FILE2 open2 (char *filename);
```

Implementado e funcionando. Procura pelo arquivo indicado pelo filename. Se existe, ele é adicionado na estrutura de arquivos abertos e é retornado um handle, que pode ser usado posteriormente por outras funções para manipular este arquivo.

```
int close2 (FILE2 handle);
```

Implementado e funcionando. Recebe um handle, procura este handle na estrutura de arquivos aberto e o fecha, removendo da lista de arquivos abertos.

```
int read2 (FILE2 handle, char *buffer, int size);
```

Implementado e funcionando. É copiado a quantidade de bytes indicada por size do arquivo indicado por handle para o buffer. Se o tamanho do arquivo for menor que size, é retornado o tamanho do arquivo, caso contrário é retornado o próprio size.

```
int write2 (FILE2 handle, char *buffer, int size);
```

Implementado e funcionando parcialmente. É possível gravar dados em cima dos dados existentes do arquivo. Por exemplo, alterar uma string com novo conteúdo, como será demonstrado nos testes.

```
int truncate2 (FILE2 handle);
```

Implementado e funcionando. Vai truncar o arquivo a partir do byte onde o ponteiro do arquivo está posicionado. O ponteiro deve ser posicionado através da função seek2.

```
int seek2 (FILE2 handle, DWORD offset);
```

Implementado e funcionando. Coloca o ponteiro do arquivo indicado pelo handle no byte indicado pelo offset.

```
int mkdir2 (char *pathname);
```

Implementado e funcionando. Procura o caminho indicado por pathname. Se o caminho já existe, é criado um diretório ali, alocando um record com os dados correspondentes e atualizando os bitmaps de blocos e inodes.

```
int rmdir2 (char *pathname);
```

Implementado e funcionando. Procura pelo diretório indicado pelo pathname. Se existe, seus inode/record são marcados como inválidos, e os bitmaps de blocos e inodes tem os bits referente ao arquivo marcados como livres.

```
DIR2 opendir2 (char *pathname);
```

Implementado e funcionando. Procura pelo diretório indicado pelo pathname. Se existe, ele é adicionado na estrutura de arquivos abertos e é retornado um handle, que pode ser usado posteriormente por outras funções para manipular este diretório.

```
int readdir2 (DIR2 handle, DIRENT2 *dentry);
```

Implementado e funcionando. Procura pelo diretório indicado pelo handle. Se existe, os dados adequados do primeiro record encontrado no diretório é retornado através de dentry. Cada chamada posterior ira ler o record seguinte. Ao chegar no fim do diretório, a chamada de readdir2 retornará um erro dizendo que não existem mais records disponíveis. A chamada seguinte, após o erro, começará a ler a partir do primeiro record novamente.

```
int closedir2 (DIR2 handle);
```

Implementado e funcionando. Procura pelo diretório indicado pelo handle na estrutura de arquivos aberto e o fecha, removendo da lista de arquivos abertos.

Para o caso de não estarem funcionando adequadamente, descrever qual é a sua visão do por que desse não funcionamento.

Uma grande limitação da implementação é que só está funcionando o bloco direto, apontado pelo ponteiro direto do inode. Não está funcinoando a indireção simples, nem indireção dupla. Um erro no

início do planejamento de como o trabalho seria feito acarretou que a maior parte do projeto teve de ser refeito. Inicialmente, as funções estavam sendo implementadas com uma finalidade explícita, ou seja, cada função (opendir2, readdir2...) seria implementada de forma independente, do começo ao fim, realizando a tarefa desejada. Ficou muito claro que este não era o melhor caminho e que muitas operações seriam repetidas e poderiam ser compartilhadas entre as funções. A partir disso, funções genéricas começaram a ser implementadas. Coisas como fornecer o número de um inode, e recuperar os blocos que ele aponta. Passar o número de um bloco e procurar por um record, podendo filtrar por tipo (diretório, arquivo, inválido) foram implementadas e reaproveitadas em praticamente todas funções. Essas funções auxiliares serviram como base para implementação do trabalho como um todo.

Descreva os testes realizados pelo grupo e se o resultado esperado se concretizou. Cada programa de teste elaborado e entregue pelo grupo deve ter uma descricao de seu funcionamento, quais as entradas fornecidas e quais os resultados finais esperados.

Foi desenvolvido quatro testes. Eles foram pensados em sequência, onde um executa após o outro, já que um teste pode depender de um arquivo que foi criado no teste anterior.

No ínicio e fim de cada teste, uma função de debug é chamada para mostrar os dados existentes no bloco 0 (raiz), bloco 2 (/sub/), setor de inodes (setor 3) e inodes e blocos ocupados no bitmap, facilitando então visualizar como o disco estava antes e após a realização do teste, ficando evidente as alterações feitas.

Observar: no debug do disco após o teste, é possível visualizar todos arquivos criados, alterados, removidos e também os bits ocupados do bitmap. Durante a execução de cada função, os principais passos sendo executados são mostrados na CLI através de *printf*s.

Teste1:

1. São criados 17 arquivos na raiz, nomeados de /file, /file0, /file1, /file2... usando a função create2.
2. São criados 3 diretórios na raiz, dir1, dir2, dir3 e um diretório dentro de /sub, /sub/dir.
3. Através das funções opendir2/open2, são abertos 23 arquivos, onde nos ultimos 3 são apresentados erros, já que o limite de arquivos abertos é 20.

Teste2:

1. Através das funções opendir2/open2 é aberto o diretório /sub/ e os arquivos /file, /arq e /sub/arq2.
2. Através da função read2 é realizado a leitura dos arquivos arq e arq2. Primeiro é feita uma requisação de leitura de 80 bytes e depois 10 bytes para o arquivo arq, em seguida é feita a leitura de 10 bytes e depois 80 bytes do arquivo arq2.

3. Através das funções close2/closedir2, os arquivos abertos são fechados.

Teste3:

1. Através da função open2, o arquivo /arq é aberto.
2. Usando uma combinação das funções seek2, truncate2 e read2, o arquivo /arq é truncado de 5 em 5 bytes e a conteúdo resultado é mostrado.

Teste4:

1. Através da função create2 os arquivos /sub/file3, /sub/file4, /sub/file5 e /sub/file6 são criados
2. Através das funções opendir2 e readdir2, o diretório /sub/ é aberto e lido record a record para saber o conteúdo existente no disco.
3. Através das funçãos open2 e read2, o arquivo /sub/arq2 é aberto e lido.
4. Através das funçãos seek2 e write2, a string "**ALTERANDO**"é primeiramente escrita a partir do byte 25 de arq2, em seguida a partir do byte 5.
5. O arquivo arq2 é fechado usando a função close2
6. Através das funções delete2 e rmdir2, os arquivos /sub/file3, /sub/file4, /sub/file5 e /dir1 são removidos

Todos os testes tem os resultados esperados.

Com certeza existem casos a serem estudados onde os testes falhariam, necessitando então de implementação para tratar os mesmos. Fiz a suposição do usuário perfeito, assumindo que todas chamadas de função vão passar entradas condizentes com as esperadas.

Quais as principais dificuldades encontradas no desenvolvimento deste trabalho e quais as soluções empregadas para contorná-las?

As principais dificuldades trouxeram as maiores recompensas. É raro um trabalho de uma disciplina ser complexo, exigir do aluno, e ao mesmo tempo ser prazeroso de fazer. O trabalho como ferramenta de aprendizado foi muito válido para entender detalhadamente (no âmbito da disciplina) o funcionamento de records, inodes, setores/blocos e todas as implicações dessas coisas em um sistema de arquivos. Em outras disciplinas o trabalho normalmente é algo mecânico e repetitivo, diferente do T2FS onde a fixação e entendimento do que se está fazendo é extremamente importante, caso contrário é impossível fazer!

Passando a parte de entender a teoria em si do trabalho, as dificuldades maiores ficaram por conta da programação em si, em como resolver cada problema. Como já citado, o planejamento errado no ínicio do desenvolvimento do projeto atrasou bastante. Fazer um mapa de tudo que precisa ser feito em cada função e projetar funções auxiliares genéricas foi a forma de contornar este problema. A maior dificuldade está em resolver todos casos de warnings do compilador. Duas warnings basicamente se repetem diversas vezes em todo código. Apesar de funcionar, warnings não estão lá sem motivo e resolvi todas que foram possíveis.

O desenvolvimento do trabalho foi documentado e está disponível em `https://github.com/gumadeiras/inf-sistemas-operacionais-I`.