Trabalho Prático II – ENTREGA: 25 de JUNHO de 2015

Implementação de um Sistema de Arquivos

1 Descrição Geral

O objetivo deste trabalho é a aplicação dos conceitos de sistemas operacionais na implementação de um Sistema de Arquivos que empregue alocação indexada para a implementação de arquivos e diretórios.

Esse Sistema de Arquivos será chamado, daqui para diante, de T2FS (*Task 2 – File System – Versão 2015.1*) e deverá ser implementado, OBRIGATORIAMENTE, na linguagem "C" sem o uso de outras bibliotecas, com exceção da *libc* e da *libapidisk* (subsistema de E/S, que será fornecida pelo professor). Além disso, a implementação deverá executar em ambiente Unix, usando a máquina virtual empregada nos exercícios práticos.

O sistema de arquivos T2FS deverá ser fornecido na forma de um arquivo de biblioteca chamado *libt2fs.a*. Essa biblioteca fornecerá uma interface de programação através da qual programas de usuário e utilitários – escritos em C – possam interagir com o sistema de arquivos. Comandos típicos aplicados sobre arquivos são o *create*, *open*, *read*, *write*, *close*, etc.

A figura 1 ilustra os componentes deste trabalho. Notar a existência de três camadas de software. A camada superior é composta por programas de usuários, tais como os programas de teste escritos pelo professor e por vocês mesmos, e por programas utilitários do sistema.

A camada intermediária representa o Sistema de Arquivos T2FS. A implementação dessa camada é sua responsabilidade e o principal objetivo deste trabalho.

Por fim, a camada inferior, que representa o acesso ao disco, é implementada pela *apidisk*, que será fornecida junto com a especificação deste trabalho. A camada *apidisk* emula o driver de dispositivo disco rígido e o próprio disco rígido. Essa camada é composta por um arquivo que simulará um disco lógico formatado em T2FS, e por funções básicas de leitura e escrita de **setores lógicos** nesse disco. As funções básicas de leitura e escrita simulam as solicitações enviadas ao driver de dispositivo (disco T2FS).

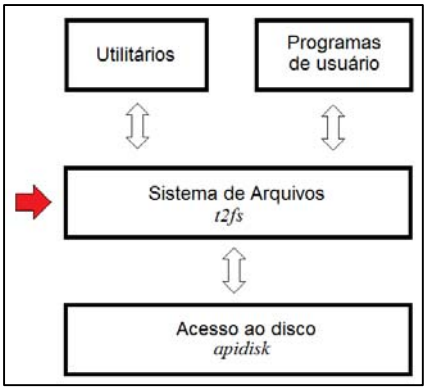

Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

1.1 Setores físicos, lógicos e blocos lógicos

Os discos rígidos são compostos por uma controladora e uma parte mecânica, da qual fazem parte a mídia magnética (pratos) e o conjunto de braços e cabeçotes de leitura e escrita. O disco físico pode ser visto como uma estrutura tridimensional composta pela superfície do prato (cabeçote), por cilindros (trilhas concêntricas) que, por sua vez, são divididos em **setores físicos** com um número fixo de bytes.

A tarefa da controladora de disco é transformar a estrutura tridimensional (*Cylinder, Head, Sector* – CHS) em uma sequência linear de **setores lógicos** com o mesmo tamanho dos setores físicos. Esse procedimento é conhecido como *Linear Block Address* (LBA) Os setores lógicos também são denominados de blocos físicos. Os setores lógicos são numerados de 0 até S−1, onde S é o número total de setores lógicos do disco e são agrupados, segundo o formato do sistema de arquivos, para formar os **blocos lógicos** (ou *cluster*, na terminologia da Microsoft).

Na formatação física, os setores físicos contêm 256 bytes e, por consequência, os setores lógicos também. Ao se formatar logicamente o disco para o sistema de arquivos T2FS, os setores lógicos serão agrupados para formar os blocos lógicos do T2FS. Dessa forma, um bloco lógico T2FS é formado por uma sequência contígua de *n* setores lógicos. A figura 2 ilustra esses conceitos.

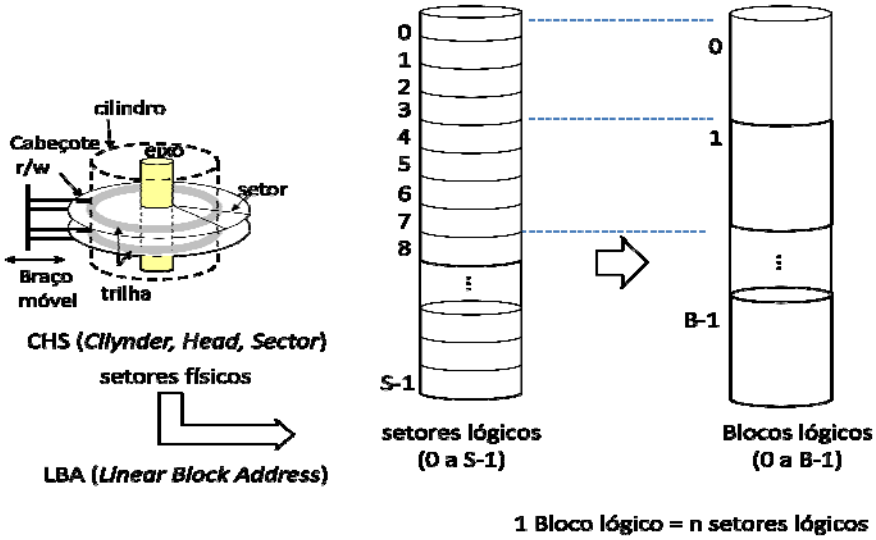

Figura 2 – setores físicos, setores lógicos e blocos lógicos

2 Estrutura de uma partição T2FS

Uma partição – ou volume – T2FS é composta por cinco áreas sucessivas: superbloco, *bitmap* de blocos livres e ocupados; *bitmap* de *i-nodes* livres e ocupados; *i-nodes* T2FS e blocos de dados.

- Superbloco (bloco 0): fornece as informações relativas à organização do sistema de arquivos T2FS;
- *Bitmap* de blocos livres e ocupados (a partir do bloco 1): Ocupa *i* blocos do disco, usados para controlar a alocação dos blocos lógicos;
- *Bitmap* de *i-nodes* (a partir do bloco *i+1*): Ocupa *j* blocos do disco, usados para controlar a alocação dos *i-nodes* livres e ocupados;
- *I-nodes* T2FS (a partir do bloco *i+j+1*): Ocupa *k* blocos do disco. Nestes blocos estão armazenados os *i-nodes* do T2FS.
- Blocos de dados (a partir do bloco *i+j+k+1*): onde serão colocados os dados dos arquivos. Ocupa o restante do disco até o bloco lógico *B-1*

A quantidade de blocos ocupados pelos *bitmaps*, assim como pelos *i-nodes*, depende do tamanho da partição do T2FS e do tamanho do setor lógico em bytes. Essas informações estão no superbloco e especificam a geometria (formato) do disco T2F2. A figura 3 ilustra a organização de um disco T2FS. As diferentes estruturas que formam um disco lógico T2FS são descritas na sequência.

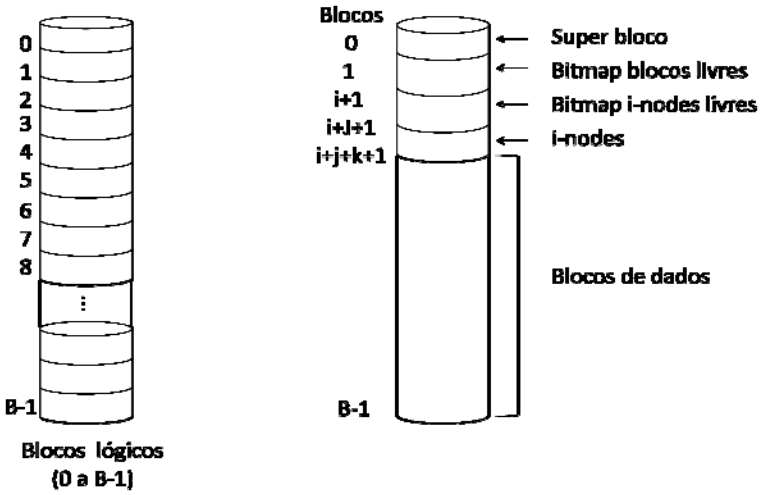

Figura 3 – Estrutura de uma partição T2FS

2.1 Superbloco

O objetivo do superbloco é manter informações que descrevem a organização da partição para uso do sistema operacional, fornecendo dados de identificação, localização das áreas onde estão armazenados as estruturas de controle (*bitmaps*, *i-nodes*, diretório raiz, etc), a quantidade e tamanho de blocos, tamanho total da partição, entre outros. No caso específico deste trabalho, o superbloco do T2FS ocupa o primeiro bloco lógico do disco, conforme definido na tabela 1, onde todos os valores numéricos são armazenados em formato *little-endian*. Apesar de ocupar um bloco lógico, todas as informações relativas à geometria do disco encontram-se no primeiro setor lógico desse bloco.

Posição relativa	Tamanho (bytes)	Nome	Valor	Descrição
0	4	*Id*	0x54 0x32 0x46 0x53	Identificação do sistema de arquivo. É formado pelas letras "T2FS".
4	2	*Version*	0x0F1 0x7D	Versão atual desse sistema de arquivos: (valor fixo 0x7DF=2015; 1=1 semestre).
6	2	*SuperBlockSize*	0x01 0x00	Quantidade de blocos que formam o superbloco. (1 bloco).
8	4	*DiskSize*		Tamanho total, em bytes, da partição T2FS. Inclui o superbloco, os dois *bitmaps*, *i-nodes* e os blocos de dados. Por exemplo, se esses bytes tiverem os valores hexadecimais 0x00 0x01 0x10 0x00, então a partição terá 1.048.832 bytes.
12	4	*NofBlocks*		Quantidade total de blocos de dados na partição T2FS. Inclui o superbloco, *bitmaps*, *i-nodes* e os blocos de dados. Por exemplo, se esses bytes tiverem os valores hexadecimais 0x00 0x04 0x00 0x00, então a partição terá 1024 blocos de dados.
16	4	*BlockSize*		Número de bytes de um bloco. Por exemplo, se esses bytes tiverem os valores hexadecimais 0x00 0x08 0x00 0x00, então um bloco de dados terá 2048 bytes.
20	4	*BitmapBlockS*		Primeiro bloco lógico do *bitmap* de blocos livres e ocupados
24	4	*BitmapInodeS*		Primeiro bloco lógico do *bitmap* de *i-nodes* livres e ocupados.
28	4	*I-nodeBlock*		Primeiro bloco lógico da área de *i-nodes*
32	4	*FirstDataBlock*		Primeiro bloco lógico da área de blocos de dados.
36-Final		*Reservado*		Não usado

Tabela 1 – Campos do superbloco T2FS (bloco lógico)

2.2 *Bitmaps* de blocos livres e ocupados

Os blocos, iniciando com o bloco indicado por *BitmapBlockS* do Superbloco, são destinados a armazenar o *bitmap* que controlam quais blocos de dados estão livres e ocupados. O tamanho exato, em blocos, desse *bitmap* dependerá do tamanho do bloco e da partição T2FS.

2.3 *Bitmaps* de *i-nodes* livres e ocupados

Os blocos, iniciando com o bloco indicado por *BitmapInodeS* do Superbloco, são destinados à armazenar o *bitmap* que controla a alocação dos blocos usados para os *i-nodes*. O tamanho exato, em blocos, desse *bitmap* dependerá do tamanho do bloco e da partição T2FS.

2.4 *I-nodes* T2FS

Os blocos, iniciando com o bloco *I-nodeBlock* do Superbloco, são destinados para a área de *i-nodes*. Portanto, o T2FS emprega um sistema de alocação indexada, onde cada *i-node* fornece o endereços lógicos dos blocos de dados que formam um determinado arquivo. Cada *i-node* T2FS tem um tamanho de 64 bytes. Sua estrutura interna é descrita na seção 2.8.

2.5 Blocos de Dados

Os blocos, iniciando com o bloco *FirstDataBlock* do Superbloco, são destinados para os blocos de dados. Esses blocos podem armazenar dados de arquivos regulares, dados de arquivos de diretório (registros de conteúdo do diretório) e índices de blocos de dados que formam um arquivo, quando for necessário usar indireção.

Um arquivo T2FS é composto por um conjunto de blocos lógicos T2FS. Convém ressaltar que um bloco lógico é a menor alocação possível, ou seja, um arquivo de 1 byte consome um bloco lógico inteiro. Para efeitos de atributos do arquivo, o tamanho do mesmo será 1 byte, mas, internamente, ele consumirá um bloco lógico inteiro.

2.6 Implementação de arquivos no T2FS

O T2FS é um sistema de arquivos que emprega o método de alocação indexada com ponteiros diretos e indiretos, sendo que o disco lógico possui a estrutura apresentada na figura 3. A descrição da geometria do disco, isso é, número de blocos lógicos que compõe cada elemento dessa estrutura é fornecida pelo primeiro bloco lógico do disco, o bloco zero, que corresponde ao superbloco. O superbloco sempre terá o tamanho de um bloco lógico com os campos descritos na tabela 1.

O diretório T2FS é um arquivo linear formado por um registro especial (seção 2.9) de tamanho fixo. Os diretórios e arquivos do T2FS seguem uma estrutura hierárquica do tipo árvore.

Os nomes simbólicos para os arquivos e diretórios do T2FS têm no máximo 31 caracteres alfanuméricos. Isso significa que não estão autorizados caracteres especiais nem espaços em branco em nomes T2FS. Não há obrigatoriedade de nenhuma forma de extensão. Os nomes são *case-sensitive*.

2.7 Gerência do espaço em disco de blocos de dados e de *i-nodes*

A determinação dos blocos de dados (e *i-nodes*) livres ou ocupados é feita através da análise da área de *bitmap*, onde cada bit indica a alocação de um bloco de dados do disco (ou *i-node*). O bit "0" do primeiro byte desse arquivo indica a alocação do bloco de dados (ou *i-node*) "0"; o bit "1" desse primeiro byte indica a alocação do bloco de dados (ou *i-node*) "1"; e assim por diante, até o último bloco de dados (ou *i-node*). Se o valor do bit for "0", então o bloco de dados (ou *i-node*) correspondente está livre; se for "1", então o bloco de dados (ou *i-node*) correspondente está ocupado.

Observação: em um disco formatado em T2FS os primeiros blocos lógicos sempre estarão ocupados, pois são aqueles utilizados pelo superbloco, pelos próprios *bitmaps* e pelos *i-nodes*.

2.8 *I-nodes* T2FS

Os *i-nodes* T2FS são registros de 64 bytes, organizados conforme a tabela 2.

Posição relativa	Tamanho (bytes)	Nome	Descrição
0	40	*dataPtr[10]*	Ponteiros diretos para blocos de dados do arquivo.
40	4	*singleIndPtr*	Ponteiro de indireção simples. Apontam para um bloco de dados composto por índices, que são os ponteiros para os blocos de dados do arquivo.
44	4	*doubleIndPtr*	Ponteiro de indireção dupla. Apontam para um bloco de dados composto por índices, que são ponteiro para outros blocos de índices. Esses últimos são ponteiro para os blocos de dados do arquivo.
48	16	*Reservados*	Não usados.

Tabela 2 – Estrutura interna de um *i-node* (estrutura *t2fs_inode*)

A quantidade de blocos lógicos que formam a área de *i-nodes* depende da quantidade de blocos lógicos da partição T2FS. Os *i-nodes* são numerados desde zero até o número de blocos de dados previsto na estrutura da partição T2FS descrita no superbloco, menos 1.

No T2FS, cada arquivo possui um *i-node* associado, que fornece o endereço dos blocos de dados que formam esse arquivo. O *i-node* 0 (zero) é reservado para o diretório raiz (“/”)

Notar que a ocupação dos ponteiros é feita em ordem, iniciando pelos ponteiros diretos e terminando pelo ponteiro de mais alto nível de indireção. Caso os ponteiros não sejam usados, esses devem receber o valor 0x0FFFFFFF (o que qualifica-o como inválido).

2.9 Implementação de diretórios no T2FS

Os diretórios T2FS seguem uma organização em árvore, ou seja, dentro de um diretório é possível definir um subdiretório, e assim sucessivamente. Portanto, um diretório T2FS pode conter registros de:

- arquivos regulares;
- arquivos de diretórios (subdiretórios).

Os arquivos no T2FS podem ser especificados de duas formas: absoluta ou relativa. Na forma absoluta, deve-se informar o caminho desde o diretório raiz; na forma relativa, pode-se indicar o caminho a partir do diretório corrente de trabalho. O caractere de barra (“/”) será utilizado na formação dos caminhos absolutos e relativos. Qualquer caminho que inicie com esse caractere deverá ser interpretado como caminho absoluto, caso contrário, como relativo. A notação a ser empregada para descrever os caminhos é a mesma do Unix, isso é, o caractere “.” indica o diretório corrente, e a sequência de caracteres “..” indica o diretório-pai.

Cada arquivo existente em um disco formatado em T2FS possui uma entrada (registro) em um diretório. Notar que todos os subdiretórios devem possuir duas entradas de diretório: a entrada “.” (ponto) e a entrada “..” (ponto ponto), para indicar o próprio diretório e o seu diretório pai, respectivamente.Os diretórios são implementados através de arquivos organizados internamente como uma lista linear de registros de tamanho fixo. Cada registro é uma entrada do diretório e está associada a um arquivo (regular ou subdiretório). A tabela 3 mostra a estrutura de um registro (estrutura *t2fs_record*), onde todos os valores numéricos estão armazenados em formato *little-endian*.

Posição relativa	Tamanho (bytes)	Nome	Descrição
0	1	*TypeVal*	Tipo da entrada. Indica se o registro é válido e, se for, o tipo do arquivo (regular ou diretório). • 0x00FF, registro inválido (não associado a nenhum arquivos); • 0x01 arquivo regular; • 0x02, arquivo de diretório.
1	31	*name*	Nome do arquivo: *string* com caracteres ASCII (0x21 até 0x7A), *case sensitive*. O *string* deve terminar com o caractere especial "\0" (0x00).
32	4	*blocksFileSize*	Tamanho do arquivo. Expresso, apenas, em número de blocos de dados (não estão inclusos eventuais blocos de índice).
36	4	*bytesFileSize*	Tamanho do arquivo. Expresso em número de bytes. Notar que o tamanho de um arquivo não é um múltiplo do tamanho dos blocos de dados. Portanto, o último bloco de dados pode não estar totalmente utilizado. Assim, a detecção do término do arquivo dependerá da observação desse campo do registro.
40	4	*i-node*	Endereço do *i-node* que lista os blocos de dados que formam o arquivo.
44	20	*Reservado*	Não utilizado

Tabela 3 – Estrutura interna de uma entrada de diretório no T2FS (estrutura *t2fs_record*)

2.10 Formato dos Blocos de Índice

Os blocos de índice são blocos lógicos do disco onde estão armazenados ponteiros para blocos de dados ou para outros blocos de índice (indireções), de um determinado arquivo. Cada ponteiro desses blocos de índice ocupa 4 (quatro) bytes e está armazenado em formato *little endian*.

A ocupação dos ponteiros deve ser feita em ordem, iniciando na posição 0 (zero) do bloco. Os ponteiros não usados devem receber o valor 0x0FFFFFFF. O término da lista de ponteiros é identificado por encontrar um ponteiro inválido ou por ter lido o último ponteiro do último bloco de índice.

3 Interface de Programação da T2FS (libt2fs.a)

Sua tarefa é implementar a biblioteca *libt2fs.a*, que possibilitará o acesso aos recursos do sistema de arquivos T2FS. Esses recursos podem ser arquivos regulares ou diretórios.

As funções a serem implementadas estão resumidas na tabela 4 e são detalhadas a seguir. Na tabela 4 são usados alguns tipos de dados e protótipos de função que estão definidos no arquivo *t2fs.h* fornecido junto com a especificação deste trabalho. Os protótipos de função e os tipos de dados existentes nesse arquivo de inclusão (*header files* ou arquivo *.h*) deve seguir, rigorosamente, o especificado neste documento.

A implementação do sistema de arquivos T2FS deve ser feita de tal forma que seja possível ter-se até 20 (vinte) arquivos abertos simultaneamente.

Nome	Descrição
`int identify2 (char *name, int size)`	Informa a identificação dos desenvolvedores do T2FS.
`FILE2 create2 (char *filename)`	Função usada para criar um novo arquivo no disco.
`int delete2 (char *filename)`	Função usada para remover (apagar) um arquivo do disco.
`FILE2 open2 (char *filename)`	Função que abre um arquivo existente no disco.
`int close2 (FILE2 handle)`	Função usada para fechar um arquivo.
`int read2 (FILE2 handle, char *buffer, int size)`	Função usada para realizar a leitura de uma certa quantidade de bytes (*size*) de um arquivo.
`int write2 (FILE2 handle, char *buffer, int size)`	Função usada para realizar a escrita de uma certa quantidade de bytes (*size*) de um arquivo.
`int seek2 (FILE2 handle, unsigned int offset)`	Altera o contador de posição (*current pointer*) do arquivo.
`int mkdir2 (char *pathname)`	Função usada para criar um novo diretório.
`int rmdir2 (char *pathname)`	Função usada para remover (apagar) um diretório do disco.
`DIR2 opendir2 (char *pathname)`	Função que abre um diretório existente no disco.
`int readdir2 (DIR2 handle, DIRENT *dentry)`	Função usada para ler as entradas de um diretório.
`int closedir2 (DIR2 handle)`	Função usada para fechar um arquivo.
`int chdir2 (char *pathname)`	Função usada para alterar o diretório corrente.
`int getcwd2 (char *pathname, int size)`	Função usada para obter o caminho do diretório corrente.

Tabela 4 – Interface de programação de aplicações – API - da *libt2fs*

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

Função usada para identificar os desenvolvedores do T2FS. Essa função copia um string de identificação para o ponteiro indicado por "**`name`**". Essa cópia não pode exceder o tamanho do buffer, informado pelo parâmetro "**`size`**". O *string* deve ser formado apenas por caracteres ASCII (Valores entre 0x20 e 0x7A) e terminado por '\0' com o nome e número do cartão dos participantes do grupo.

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Em caso de erro, será retornado um valor diferente de zero.

`FILE2 create2 (char *filename)`

Função que cria um novo arquivo. O nome desse novo arquivo é aquele informado pelo parâmetro "**`filename`**". O contador de posição do arquivo (*current pointer*) deve ser colocado na posição zero. Além disso, caso já exista um arquivo ou diretório com o mesmo nome, a função deverá retornar um erro de criação.

A função deve retornar o identificador (*handle*) do arquivo, que será usado em chamadas posteriores do sistema de arquivo para fins de manipulação do arquivo criado. Caso ocorra erro deve retornar um valor negativo.

`int delete2 (char *filename)`

Função usada para apagar um arquivo do disco. O nome do arquivo a ser apagado é aquele informado pelo parâmetro "**`filename`**".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Em caso de erro, será retornado um valor diferente de zero.

`FILE2 open2 (char *filename)`

Função que abre um arquivo existente no disco. O nome desse novo arquivo é aquele informado pelo parâmetro "**`filename`**". Ao abrir um arquivo, o contador de posição do arquivo (*current pointer*) deve ser colocado na posição zero.

A função deve retornar o identificador (*handle*) do arquivo, que será usado em chamadas posteriores do sistema de arquivo para fins de manipulação do arquivo. Todos os arquivos abertos por esta chamada são abertos em leitura e em escrita. O ponto em que a leitura, ou escrita, será realizada é fornecido pelo valor *current_pointer* (ver função `seek2`).

Caso ocorra erro deve retornar um valor negativo.

`int close2 (FILE2 handle)`

Função que fecha o arquivo identificado pelo parâmetro "**`handle`**".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Em caso de erro, será retornado um valor diferente de zero.

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

Função que realiza a leitura de "**`size`**" bytes do arquivo identificado por "**`handle`**". Os bytes lidos são colocados na área apontada por "**`buffer`**".

Após realizada a leitura o contador de posição (*current pointer*) deve ser ajustado para o byte seguinte ao último lido.

Se a operação foi realizada com sucesso, a função retorna o número de bytes lidos. Se o número de bytes lidos – retornados pela função – for diferente do número de bytes solicitados para leitura (parâmetro *size*), então o contador de posição atingiu o final do arquivo, sendo essa a forma de identificar o final do arquivo.

Caso ocorra algum erro durante a execução da função, esta retornará um valor negativo.

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

Função que realiza a escrita de "**`size`**" bytes no arquivo identificado por "**`handle`**". Os bytes a serem escritos estão na área apontada por "**`buffer`**".

Após ter sido realizada a escrita o contador de posição (*current pointer*) do arquivo deve ser ajustado para o byte seguinte ao último byte escrito.

Se a operação foi realizada com sucesso, a função retorna o número de bytes efetivamente escritos. Caso ocorra algum erro durante a execução da função, esta retornará um valor negativo.

`int seek2 (FILE2 handle, unsigned int offset)`

Função usada para posicionar o contador de posições (*current pointer*) do arquivo identificado por "**`handle`**" na posição dada pelo parâmetro "**`offset`**". Esse valor corresponde ao deslocamento, em bytes, contados a partir do início do arquivo. Se o valor de "**`offset`**" for "-1", o *current_pointer* deverá ser posicionado no byte seguinte ao final do arquivo, permitindo que novos dados sejam adicionados a um arquivo já existente.

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Caso ocorra algum erro, a função retorna um valor diferente de zero.

`int mkdir2 (char *pathname)`

Função que cria um novo diretório. O caminho desse novo diretório é aquele informado pelo parâmetro "**`pathname`**", que pode ser absoluto ou relativo.

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Caso ocorra algum erro, a função retorna um valor diferente de zero. São considerados erros quaisquer situações em que o diretório não possa ser criado, incluindo a existência de um arquivo ou diretório com o mesmo "**`pathname`**".

Observar que ao ser criado um novo diretório, deverão ser criadas, automaticamente, duas entradas: "." e "..". A entrada "." corresponde ao descritor do diretório recém criado e a entrada ".." à entrada de seu diretório pai. No caso do diretório raiz, essas duas entradas apontam para o próprio descritor do diretório raiz.

`int rmdir2 (char *pathname)`

Função usada para apagar um diretório do disco. O caminho do diretório a ser apagado é aquele informado pelo parâmetro "**`pathname`**".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Em caso de erro, será retornado um valor diferente de zero. São considerados erros quaisquer situações que impeçam a operação, incluindo: (a) o diretório a ser removido não estar vazio, (b) "**`pathname`**" não existente, (c) algum dos componentes do "**`pathname`**" não existem, isso é, caminho inválido (d) o "**`pathname`**" indicado não é um arquivo ou (e) o "**`pathname`**" indica os diretórios "." ou "..".

DIR2 *opendir2* (char *pathname)

Função que abre um diretório existente no disco. O caminho desse diretório é aquele informado pelo parâmetro "**pathname**".

Se a operação foi realizada com sucesso, a função deve retornar o identificador (*handle*) do diretório e posicionar o ponteiro de entradas (*current entry*) na primeira posição válida do diretório "**pathname**". O identificador retornado será usado na chamada "***readdir2***" para a leitura das entradas desse diretório. Caso ocorra erro, a função deve retornar um valor negativo.

int *readdir2* (DIR2 handle, DIRENT* dentry)

Função que realiza a leitura das entradas do diretório identificado por "**handle**". A cada chamada da função é lida uma das entradas do diretório representado pelo identificador "**handle**". Algumas das informações dessa entrada devem ser colocadas no parâmetro "**dentry**", que é do tipo "**DIRENT2**", estrutura definida no arquivo t2fs.h da seguinte forma:

```
typedef struct {
   char name[MAX_FILE_NAME_SIZE+1];
   int fileType;
   unsigned long fileSize;
} DIRENT2;.
```

Após realizada a leitura de uma entrada, o ponteiro de entradas (*current entry*) deve ser ajustado para a próxima entrada válida, seguinte à última lida.

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Caso ocorra algum erro, a função retorna um valor diferente de zero (e o conteúdo de "**dentry**" não será válido). São considerados erros qualquer situação que impeça a realização da operação, incluindo o término das entradas válidas do diretório identificado por "**handle**".

int *closedir2* (DIR2 handle)

Função que fecha o diretório identificado pelo parâmetro "**handle**".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Em caso de erro, será retornado um valor diferente de zero.

int *chdir2* (char* pathname)

Função que altera o diretório atual de trabalho (*working directory*). O caminho desse diretório é informado no parâmetro "**pathname**".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Caso ocorra algum erro, a função retorna um valor diferente de zero. São considerados erros quaisquer situações que impeçam a alteração do diretório de trabalho, incluindo a não existência do "**pathname**" informado.

int *getcwd2* (char* pathname, int size)

Função que informa o diretório atual de trabalho. O T2FS deve copiar o *pathname* do diretório de trabalho, incluindo o '\0' do final do string, para o buffer indicado por "**pathname**". Essa cópia não pode exceder o tamanho do buffer, informado pelo parâmetro "**size**".

Se a operação foi realizada com sucesso, a função retorna "0" (zero). Caso ocorra algum erro, a função retorna um valor diferente de zero. São considerados erros quaisquer situações que impeçam a cópia do *pathname* do diretório de trabalho para "**pathname**", incluindo espaço insuficiente, conforme informado por "**size**".

4 Interface da *apidisk* (*libapidisk.o*)

Para fins deste trabalho, você receberá o binário *apidisk.o*, que realiza as operações de leitura e escrita do subsistema de E/S do disco usado pelo T2FS. Assim, o binário *apidisk.o* permitirá a leitura e a escrita dos setores lógicos do disco, que serão endereçados através de sua numeração sequencial a partir de zero. As funções dessa API estão descritas a seguir.

`int read_sector (unsigned int sector, char *buffer)`

Realiza a leitura do setor "`sector`" lógico do disco e coloca os bytes lidos no espaço de memória indicado pelo ponteiro "`buffer`".

Retorna "0", se a leitura foi realizada corretamente e um valor diferente de zero, caso tenha ocorrido algum erro.

`int write_sector (unsigned int sector, char *buffer)`

Realiza a escrita do conteúdo da memória indicada pelo ponteiro "`buffer`" no setor "`sector`" lógico do disco.

Retorna "0", se a escrita foi bem sucedida; retorna um valor diferente de zero, caso tenha ocorrido algum erro.

Por questões de simplificação, o binário *apidisk.o*, que implementa as funções *read_sector*() e *write_sector*(), e o arquivo de inclusão *apidisk.h*, com os protótipos dessas funções, serão fornecidos pelo professor. Além disso, será fornecido um arquivo de dados para emulação do disco onde estará o sistema de arquivos T2FS.

5 Geração da *libt2fs*

As funcionalidades do sistema de arquivos T2FS deverão ser disponibilizadas através de uma biblioteca de nome *libt2fs.a*. Para gerar essa biblioteca deverá ser utilizada a seguinte "receita":

1. Cada arquivo "<*file_name.c*>" deverá ser compilado com a seguinte linha de comando:

```
gcc -c <file_name.c> -Wall
```

2. Para gerar a *libt2fs.a,* todos os arquivos compilados (representados por *file_1*, *file_2*, ...) e o binário fornecido *apidisk.o* devem ser ligados usando a seguinte linha de comando:

```
ar crs libt2fs.a <file_1>.o <file_2>.o ... apidisk.o
```

Para ambas as etapas: compilação e geração da biblioteca, não deverão ocorrer mensagens de erro ou de "*warning*".

Para fins de teste, a biblioteca *libt2fs.a* deverá ser ligada com o programa chamador. Para fazer a ligação deve-se utilizar a seguinte linha de comando:

```
gcc -o <nome_exec> <nome_chamador.c> -L<lib_dir> -lt2fs -Wall
```

onde <*nome_exec*> é o nome do executável, <*nome_chamador.c*> é o nome do arquivo fonte onde é realizada a chamada das funções da biblioteca, <*lib_dir*> é o caminho do diretório onde está a bibliotecas "*libt2fs.a*".

6 Questionário

1. Nome dos componentes do grupo e número do cartão.
2. Indique, para CADA UMA das funções que formam a biblioteca *libt2fs*, se as mesmas estão funcionando corretamente ou não. Para o caso de não estarem funcionando adequadamente, descrever qual é a sua visão do por que desse não funcionamento.
3. 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 descrição de seu funcionamento, quais as entradas fornecidas e quais os resultados finais esperados.
4. Quais as principais dificuldades encontradas no desenvolvimento deste trabalho e quais as soluções empregadas para contorná-las.

7 Entregáveis: o que deve ser entregue?

Devem ser entregues:

- Todos os arquivos fonte (arquivos ".c" e ".h") que formam a biblioteca "*libt2fs*";
- Arquivo *makefile* para criar a "*libt2fs.a*".
- O arquivo "*libt2fs.a*" e
- Arquivo PDF com as respostas ao questionário.

Os arquivos devem ser entregues em um *tar.gz* (SEM arquivos *rar* ou similares), seguindo, **obrigatoriamente**, a seguinte estrutura de diretórios e arquivos:

\t2fs		
	makefile	ARQUIVO: arquivo makefile com regras para gerar a "*libt2fs*". Deve possuir uma regra "*clean*", para limpar todos os arquivos gerados.
	relatorio.pdf	ARQUIVO: arquivo PDF com as respostas do questionário.
	include	DIRETÓRIO: local onde são postos todos os arquivo ".h". Nesse diretório deve estar o "*t2fs.h*" e o "*apidisk.h*"
	src	DIRETÓRIO: local onde são postos todos os arquivos ".c" (códigos fonte) usados na implementação do T2FS.
	teste	DIRETÓRIO: local onde são armazenados todos os arquivos de programas de teste (códigos fonte) usados para testar a implementação do T2FS.
	bin	DIRETÓRIO: local onde serão postos os programas executáveis usados para testar a implementação, ou seja, os executáveis dos programas de teste.
	lib	DIRETÓRIO: local onde será gerada a biblioteca "*libt2fs.a*". (junção da "*t2fs*" com "*apidisk.o*"). O binário *apidisk.o* também será posto neste diretório.
	t2fs_disk.dat	ARQUIVO: arquivo binário que corresponde à partição T2FS (arquivo fornecido junto com a especificação)

8 Avaliação

Para que um trabalho possa ser avaliado ele deverá cumprir com as seguintes condições:

- Entrega dentro dos prazos estabelecidos;
- Obediência à especificação: formato e nome das funções, estrutura de diretórios para a entrega,
- Compilação e geração da biblioteca sem erros ou *warnings*;
- Fornecimento de todos os arquivos solicitados;
- Questionário respondido completamente!

Itens que serão avaliados e sua valoração:

- **10,0 pontos:** clareza e organização do código, programação modular, *makefiles*, arquivos de inclusão bem feitos (sem código C dentro de um include!!) e comentários adequados;
- **20,0 pontos:** resposta ao questionário e a correta associação entre a implementação e os conceitos vistos em aula;
- **70,0 pontos:** funcionamento do sistema de arquivos T2FS de acordo com a especificação. Para isso serão utilizados programas padronizados desenvolvidos pelo professor para essa verificação.

9 Data de entrega e avisos gerais – LEIA com MUITA ATENÇÃO!!!

1. Faz parte da avaliação a obediência RIGIDA aos padrões de entrega definidos na seção 7 (arquivos tar.gz, makefiles, estruturas de diretórios, etc);
2. O trabalho pode ser feito em grupos de até TRÊS alunos (grupos com mais de três alunos terão sua nota final dividida pelo número de participantes do grupo);
3. O trabalho deverá ser entregue, via ***Moodle*** da disciplina, até as 23:55:00 horas da data prevista para a entrega. Entregar um arquivo *tar.gz* conforme descrito na seção 7;
4. O trabalho deverá ser entregue até **25 de JUNHO de 2015**;
5. Admite-se a entrega do trabalho com até UMA semana de atraso. Nesse caso, o trabalho será avaliado e, da nota alcançada (de um total de 100,00 pontos) serão diminuídos 20,00 pontos pelo atraso. Não serão aceitos trabalhos entregues além dos prazos estabelecidos.

10 Observações

Recomenda-se a troca de ideias entre os alunos. Entretanto, a identificação de cópias de trabalhos acarretará na aplicação do Código Disciplina Discente e a tomada das medidas cabíveis para essa situação.

O professor da disciplina reserva-se o direito, caso necessário, de solicitar uma demonstração do programa, onde o aluno será arguido sobre o trabalho como um todo. Nesse caso, a nota final do trabalho levará em consideração o resultado da demonstração.