Trabalho Prático II – ENTREGA: 1 de JULHO de 2014

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 2014.1*) 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.

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. Nota-se 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 (comando "*dir*", por exemplo).

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. O arquivo que simula o disco lógico é análogo àqueles usados pelas máquinas virtuais, como, por exemplo, a VirtualBox. As funções básicas de leitura e escrita simulam as solicitações enviadas ao driver de dispositivo (disco T2FS).

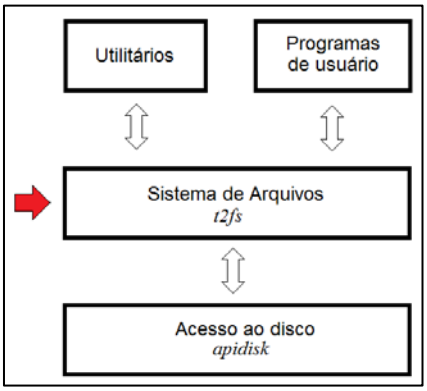

Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

2 Estrutura de uma partição T2FS

Uma partição – ou volume – T2FS é composta por duas áreas contíguas: superbloco e blocos de dados. O superbloco fornece as informações relativas à organização do sistema de arquivos T2FS. A área de blocos de dados é formada por "B" blocos, numerados de 0 a B-1. A quantidade de blocos depende do tamanho da partição do T2FS e do tamanho do bloco lógico em bytes. Essas duas áreas são detalhadas a seguir.

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 os primeiros 256 bytes do disco lógico e estão definidos na figura 2, onde todos os valores numéricos são armazenados em formato *little-endian*.

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*	0xE1 0x7D	Versão atual desse sistema de arquivos: (valor fixo 7DE=2014; 1=1 semestre).
6	2	*SuperBlockSize*	0x01 0x00	Quantidade de setore que formam o superbloco. (1 setor)
8	4	*DiskSize*	0x00 0x01 0x10 0x00	Tamanho total da partição T2FS, incluindo o tamanho do superblock. (1.048.832 bytes)
12	4	*NofBlocks*	0x00 0x04 0x00 0x00	Quantidade total de blocos de dados na partição T2FS (1024 blocos).
16	4	*BlockSize*	0x00 0x04 0x00 0x00	Tamanho de um bloco. (1024 bytes)
20	108	*Reserved*		Não usados
128	64	*BitMapReg*	Ver figura 3	Registro que descreve o arquivo que mantém o *bitmap* de blocos livres e ocupados
192	64	*RootDirReg*	Ver figura 3	Registro que descreve o arquivo que mantém as entradas do diretório raiz

Figura 2 – Campos do superbloco T2FS (256 bytes de tamanho)

2.2 Blocos de Dados

Logo após a área destinada ao superbloco T2FS iniciam os blocos de dados. Os blocos lógicos T2FS podem armazenar:

- Dados de arquivos regulares;
- Dados do arquivo de *bitmap* de blocos livres e ocupados;
- Dados de arquivos de diretório (registros do conteúdo do diretório);
- Índice dos blocos que formam um arquivo, quando for usado 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.

2.3 Implementação de arquivos no T2FS

O T2FS emprega um método de alocação indexada com ponteiros diretos e indiretos, que são campos do registro de descrição dos arquivos (entrada de diretório) descrito na seção 2.5. Para usar as indireções, é necessário criar e definir blocos de índice, que são descritos na seção 2.6.

2.4 Gerência do espaço em disco

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

O arquivo de *bitmap* possui uma estrutura igual a qualquer outro arquivo T2FS, exceto que ele não pode ser manipulado a partir de funções executadas por usuários comuns. O registro que descreve a sua localização em disco é fornecido no superbloco (campo *BitMapReg*).

2.5 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).

Por questões de simplificação, assume-se que a ÚNICA forma de acessar arquivos é através de caminhos absolutos, ou seja, na prática o T2FS não implementa caminhos relativos e não precisa ter registros de arquivos que aponte para seu próprio diretório local ou diretório pai.

Cada arquivo existente em um disco formatado em T2FS possui uma entrada (registro) em um diretório. Os diretórios são implementados através de arquivos, que são 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 figura 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). • 0xFF, registro inválido (não associado a nenhum arquivos); • 0x01 arquivo regular; • 0x02, arquivo de diretório.
1	39	*name*	Nome do arquivo. : *string* com caracteres ASCII (0x21 até 0x7A), *case sensitive*. O *string* deve terminar com o caractere especial "\0" (0x00).
40	4	*blocksFileSize*	Tamanho do arquivo. Expresso, apenas, em número de blocos de dados (não estão inclusos eventuais blocos de índice).
44	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.
48	8	*dataPtr[2]*	Dois ponteiros diretos, *dataPtr[0]* e *dataPtr[1]*, para blocos de dados do arquivo
56	4	*singleIndPtr*	Ponteiro de indireção simples, que aponta para um bloco de índices onde estão ponteiros para blocos de dados do arquivo.
60	4	*doubleIndPtr*	Ponteiro de indireção dupla, que aponta para um bloco de índices onde estão outros ponteiros para blocos de índice. Esses últimos possuem ponteiros que apontam para blocos de dados do arquivo.

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

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, estes devem receber o valor 0xFFFFFFFF (o que qualifica-o como inválido).

2.6 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 (indireção dupla), 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 0xFFFFFFFF . 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.

2.7 Resumo do T2FS (*check list*)

As principais características do T2FS, para fins de consulta rápida e verificação de conformidade da implementação com a especificação, são listadas a seguir:

- Partição T2FS é composta por um superbloco e por uma sequência de blocos de dados;
- A gerência de espaço livre no disco é feita pela técnica de *bitmap*. O *bitmap* é um arquivo especial do sistema não acessível por funções de usuário. O registro desse arquivo é fornecido no superbloco;
- Os arquivos seguem uma alocação indexada com dois níveis (indireção simples e indireção dupla);
- Os diretórios formam uma estrutura em árvore;
- O registro do diretório raiz é fornecido no superbloco;
- Os caminhos dos arquivos são sempre absolutos;
- Um arquivo de diretório é uma lista linear de registros de tamanho fixo;

3 Especificação das tarefas a serem feitas: Interface de Programação T2FS e comandos

3.1 Interface de Programação da T2FS (*libt2fs.a*)

A primeira parte da tarefa é implementar a biblioteca *libt2fs.a*, que possibilitará a leitura e escrita de arquivos no sistema de arquivos T2FS. As funções a serem implementadas estão resumidas na tabela 1 e são detalhadas a seguir.

Nome	Descrição
`char *t2fs_identify (void)`	Retorna a identificação dos desenvolvedores do T2FS.
`t2fs_file t2fs_create (char *nome)`	Função usada para criar um novo arquivo no disco.
`int t2fs_delete (char *nome)`	Função usada para remover (apagar) um arquivo do disco.
`t2fs_file t2fs_open (char *nome)`	Função que abre um arquivo existente no disco.
`int t2fs_close (t2fs_file handle)`	Função usada para fechar um arquivo.
`int t2fs_read (t2fs_file handle, char *buffer, int size)`	Função usada para realizar a leitura em um arquivo.
`int t2fs_write (t2fs_file handle, char *buffer, int size)`	Função usada para realizar a escrita em um arquivo.
`int t2fs_seek (t2fs_file handle, unsigned int offset)`	Altera o contador de posição (*current pointer*) do arquivo.

Tabela 1 – Interface de programação de aplicações – API - da *libt2fs*

Na tabela 1 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.

`char *t2fs_identify (void)`

Função usada para identificar os desenvolvedores do T2FS. Essa função deve retornar um ponteiro para um *string* 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.

`t2fs_file t2fs_create (char *nome)`

Função que cria um novo arquivo. O nome desse novo arquivo é aquele informado pelo parâmetro "**`nome`**". O contador de posição do arquivo (*current pointer*) deve ser colocado na posição zero. Além disso, caso o arquivo já exista, ele será substituído por um arquivo vazio.

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 t2fs_delete (char *nome)`

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

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

`t2fs_file` *`t2fs_open`* (`char *nome`)

Função que abre um arquivo existente no disco. O nome desse novo arquivo é aquele informado pelo parâmetro "**`nome`**". 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 `t2fs_seek`).

Caso ocorra erro deve retornar um valor negativo.

`int` *`t2fs_close`* (`t2fs_file 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` *`t2fs_read`* (`t2fs_file 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. Notar que o número de bytes lidos – retornados pela função – deve ser idêntico ao número de bytes solicitados para leitura (parâmetro *size*). Se diferentes ou, se a função retornar zero, significa que 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` *`t2fs_write`* (`t2fs_file 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 realizada a escrita o contador de posição (*current pointer*) 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` *`t2fs_seek`* (`t2fs_file 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.

3.2 Comandos de manipulação de arquivos

A segunda parte deste trabalho consiste em desenvolver alguns programas em C, usando a sua própria versão de *libt2fs.a*, que implementem comandos de manipulação de arquivos. Vocês devem implementar os seguintes comandos:

copy2t2

PROGRAMA DE IMPLEMENTAÇÂO OBRIGATÓRIA. Realiza a cópia de um arquivo qualquer do sistema de arquivos nativo da máquina para o disco lógico do T2FS. O caminho de destino no disco T2FS é absoluto. Exemplo de uso

```
%copy2t2 ./disciplina/inf01142/t2fs.pdf /aluno/sisop/trabalhopratico2.pdf
```

Copia o arquivo `t2fs.pdf`, que se encontra no sistema de arquivos nativo da máquina (diretório `disciplina`, subdiretório `inf01142`), para o disco lógico T2FS, com o nome `trabalhopratico2.pdf`. Se os diretórios de destino, `aluno` ou `sisop`, não existirem esse comando fornece uma mensagem de erro.

mkdirt2

Criação de um diretório no T2FS. O caminho a ser fornecido é absoluto. Caso algum dos diretórios fornecidos no *string* de caminho não existam, o programa deverá retornar um mensagem de erro. Exemplo de uso:

```
%mkdirt2 /semestre/disciplina/inf01142
```

Criar o diretório `inf01142` dentro do diretório `disciplina`, que por sua vez, é um subdiretório de `semestre`. Se os diretórios `disciplina` ou `semestre` não existirem esse comando fornece uma mensagem de erro.

rmdirt2

Remove um diretório. Novamente o caminho a ser fornecido é absoluto. Para que o diretório seja removido, ele deve estar vazio, ou seja, não pode haver arquivos de nenhum tipo dentro dele. Se houver arquivos, a execução do comando *rmdirt2* deverá retornar uma mensagem de erro indicativa. Da mesma forma que no mkdirt2, se algum dos diretórios fornecidos no string de caminho não existirem, o programa deverá retornar um mensagem de erro. Exemplo de uso:

```
%rmdirt2 /semestre/disciplina/inf01142
```

Remove o diretório `inf01142`, se ele estiver vazio. Se os diretórios `disciplina` ou `semestre` não existirem esse comando fornece uma mensagem de erro.

dirt2

Lista o conteúdo do diretório fornecido no caminho fornecido (absoluto). Se por um acaso não existir diretórios que aparecem no meio do *string* de caminho, o programa *dirt2* deverá retornar um mensagem de erro. A saída desse programa é a listagem do conteúdo dos registros do arquivo de diretório formatado para apresentar, na ordem, nome do arquivo, indicação se é um arquivo regular (r) ou diretório (d), a quantidade de blocos que utiliza e o tamanho do arquivo em bytes. Exemplos de uso:

```
%dirt2 /semestre/disciplina/inf01142
Aula01.pdf    r      10     19500 bytes
Atividade1    d      1        704 bytes
```

Se o diretório não existir, uma mensagem de erro deverá ser gerada.

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

Como visto em aula, o sistema de arquivos é organizado em uma arquitetura em camadas onde a camada mais baixa é hardware do disco físico propriamente dito. O disco físico, por sua vez, é composto por uma controladora de disco e por uma parte mecânica que possui a mídia magnética (pratos) e 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 essa estrutura tridimensional (*Cylinder, Head, Sector* – CHS) em uma sequência linear de **setores lógicos**, numerados de 0 até S−1, onde S é o número total de setores lógicos do disco. Esses setores lógicos são agrupados, segundo o formato do sistema de arquivos, para formar os blocos lógicos (ou *cluster*, na terminologia da Microsoft).

Na formatação física, um disco é organizado em setores com 256 bytes. Ao se formatar logicamente o disco para o sistema de arquivos T2FS, os setores lógicos são agrupados para formar o bloco lógico do T2FS. Dessa forma, um bloco lógico T2FS é uma sequência contígua de *n* setores lógicos. Cabe a camada de driver de dispositivo receber comandos de leitura e de escrita em um setor lógico *S*.

Neste trabalho você receberá pronto o binário *apidisk* que realiza as operações básicas de leitura e escrita do subsistema de E/S do disco usado pelo T2FSAssim, o binário *apidisk* permitirá a leitura e a escrita dos setores lógicos do disco, que serão endereçados através de sua numeração sequencial. Essas funções são 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*, que simula o driver de disco T2F será disponibilizado através de uma biblioteca. A biblioteca *libapidisk.a*, 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.

IMPORTANTE!

Estão sendo fornecidas duas bibliotecas: uma compilada para 32 bits e outra para 64 bits. Elas são chamadas *libapidisk32.a* e *libapidisk64.a*, respectivamente. Escolha o arquivo mais apropriada para o seu sistema de trabalho e altere o nome do mesmo para *libapidisk.a*, de maneira a compilar seu trabalho junto com a biblioteca, usando os comandos mostrados na seção 5.

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 a biblioteca *libapidisk.a* devem ser ligados usando a seguinte linha de comando:

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

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. Sem alterar a quantidade de ponteiros de alocação indexada, quais outros fatores influenciam no maior tamanho de arquivo T2FS possível? Como esse fatores influenciam nesse tamanho?

2. Supondo que você desejasse melhorar o T2FS, permitindo a criação de vínculos estritos (*hardlinks*). Que alterações seriam necessárias no T2FS? Há necessidade da criação de novas funções? Se sim, quais? Se não, porque não.

3. As estruturas de controle do T2FS contêm informações que permitem verificar a consistência de alguns de seus elementos. Isso é possível graças a um nível de redundância de informação (por exemplo, no registro de arquivo, nas entradas do diretório, o número total de blocos usados por um arquivo e o tamanho do arquivo – em bytes – permitem uma verificação). Identifique quais outros elementos são redundantes e discuta como seria possível usar essa redundância para aumentar a confiabilidade do T2FS.

4. Como você implementou a atribuição dos identificadores de arquivos (*file handler*) pelas funções *t2fs_create* e *t2fs_open*? Discuta a questão da reutilização dos mesmos.

5. Como você implementou a gerência do contador de posição (*current pointer*) usado pela função *t2fs_seek*?

6. A escrita em um arquivo (realizada pela função *t2fs_write*) requer uma sequência de leituras e escritas de blocos de dados e de blocos de controle. Qual é a sequência usada por essa função? Se essa sequência for interrompida (por falta de energia, por exemplo) entre duas operações de escrita de bloco, qual será o efeito na consistência dos dados no disco? É possível projetar uma sequência de escritas no disco que minimize a eventual perda de dados?

7. Algumas estruturas gravadas no disco são mais facilmente manipuláveis se estiverem na memória principal (como se fosse uma *cache*). Por outro lado, isso aumenta a possibilidade de perda de dados, pois as informações existentes nessa cache e que não foram escritas no disco, podem ser perdidas, caso ocorra alguma interrupção de operação do sistema. Quais informações do disco você está mantendo (e gerenciando) na memória principal e porque você as escolheu? Qual a política que você usou para decidir quando escrevê-las no disco?

8. Todas as funções implementadas funcionam corretamente? Relate, para cada uma das funções desenvolvidas, como elas foram testadas?

9. Relate as suas maiores dificuldades no desenvolvimento deste trabalho e como elas foram contornadas.

7 Entregáveis: o que deve ser entregue?

Devem ser entregues:

- Todos os arquivos fonte (arquivos ".c" e ".h") que formam a biblioteca "*libt2fs*" e os utilitários;
- Arquivo *makefile* para criar a "*libt2fs.a*" e os programas utilitários.
- O arquivo "*libt2fs.a*" e
- Arquivo PDF com as respostas do 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*" e os arquivos executáveis dos comandos *mkdirt2*, *rmdirt2*, *dirt2* e *copy2t2*. 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 e dos comandos *mkdirt2*, *rmdirt2*, *dirt2* e *copy2t2*.
	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 comandos *mkdirt2*, *rmdirt2*, *dirt2* e *copy2t2* e os programas de teste.
	lib	DIRETÓRIO: local onde será gerada a biblioteca "*libt2fs.a*". (junção da "*t2fs*" com a "*libapidisk.a*"). A biblioteca *libapidisk.a* também será posta neste diretório.
	t2fs_disk.dat	ARQUIVO: arquivo binário que corresponde à partição T2FS (arquivo fornecido junto com a especificação)

No relatório da implementação devem estar presentes (1) a identificação dos componentes do grupo (número de cartão e nome completo) e (2) as respostas ao questionário proposto (seção 6).

O trabalho deverá ser entregue até a **data prevista (1 de JULHO de 2014)**. 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.

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;
- Relatório completo!

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;
- **30,0 pontos:** resposta ao questionário e a correta associação entre a implementação e os conceitos vistos em aula;
- **60,0 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 até as 23:55:00 horas do dia 1 de JUlHO via *moodle*. Entregar um arquivo *tar.gz* conforme descrito na seção 7;
4. Trabalhos entregues atrasados serão penalizados com desconto: entrega até 8 de JULHO de 2014 (23:55:00 horas) passa a valer no máximo 80,0 PONTOS. Esse será o atraso máximo permitido (uma semana). Expirado o atraso máximo, nenhum trabalho será aceito.

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.