Trabalho Prático II – ENTREGA: 21 de JUNHO de 2016

Implementação de um Sistema de Arquivos T2FS

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 criação de arquivos e diretórios.

Esse Sistema de Arquivos será chamado, daqui para diante, de T2FS (*Task 2 – File System – Versão 2016.1*) e deverá ser implementado, OBRIGATORIAMENTE, na linguagem "C", sem o uso de outras bibliotecas, com exceção da *libc*. Além disso, a implementação deverá executar na máquina virtual fornecida no Moodle.

O sistema de arquivos T2FS deverá ser disponibilizado 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 – poderão interagir com o sistema de arquivos.

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 ou 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 do disco rígido e o próprio disco rígido. Essa camada é composta por um arquivo que simulará um disco formatado em T2FS, e por funções básicas de leitura e escrita de **setores lógicos** desse disco. As funções básicas de leitura e escrita simulam as solicitações enviadas ao *driver* de dispositivo (disco T2FS).

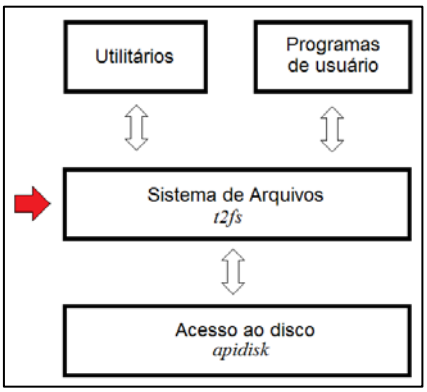

Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

1.1 Setores físicos, setores 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).

Assim, na formatação física, os setores físicos contêm, dependendo da mídia, 256, 512, 1024 ou 2048 bytes e, por consequência, os setores lógicos também têm esse tamanho. No caso específico do T2F2, considera-se que os setores

físicos têm 256 bytes. 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 lógicoT2FS é formado por uma sequência contígua de *n* setores lógicos. A figura 2 ilustra esses conceitos de forma genérica.

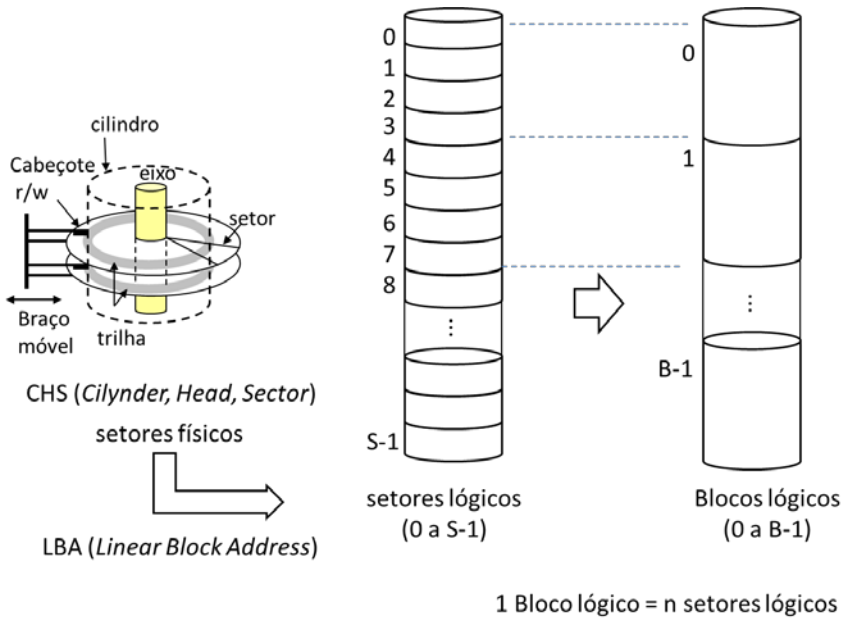

Figura 2 – setores físicos, setores lógicos e blocos lógicos (diagrama genérico)

2 Estrutura de uma partição T2FS

O espaço disponível no disco está dividido em quatro áreas contíguas: área de superbloco, área de *bitmap* (para gerência dos blocos livres e ocupados), área para o diretório raiz (*root*) e área para os blocos de índice e blocos de dados dos arquivos. A gerência do espaço em disco é feito por alocação indexada e o diretório segue uma hierarquia em árvore. Essas áreas estão representadas na figura 3 e estão detalhadas a seguir.

Área de superbloco: é a área de controle do sistema de arquivos. Essa área inicia, sempre, no **bloco lógico zero**. O superbloco contêm os seguintes elementos, na ordem em que aparecem na tabela 1. Os valores são armazenados em um formato *little-endian* (a parte menos significativa do valor é armazenada no endereço mais baixo de memória).

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*	0x01 0x7E	Versão atual desse sistema de arquivos: (valor fixo 0x7E0=2016; 1=1º semestre).
6	2	*superBlockSize*	0x01 0x00	Quantidade de blocos lógicos que formam o superbloco. (1 bloco lógico).
8	2	*blockSize*		Quantidade de setores lógicos que formam um bloco lógico
10	4	*diskSize*		Quantidade de bytes da partição T2FS. Inclui o superbloco, a área de *bitmap*, o diretório raiz e a área de

				dados.
14	4	*nOfSectors*		Quantidade total de setores lógicos na partição T2FS. Inclui o superbloco, a área de *bitmap*, o diretório raiz e a área de dados.
18	4	*bitmapSize*		Quantidade de blocos lógicos usados para armazenar o *bitmap* de blocos livres e ocupados.
22	4	*rootSize*		Quantidade de blocos lógicos usados para armazenar o diretório raiz.
26	4	*nOfDirEntries*		Quantidade de entradas no diretório raiz. (inclui as entradas "." e "..")
30	4	*fileEntrySize*		Quantidade de bytes de um registro na área de diretório.
34 até o final		*reservado*		Não usados

Tabela 1 – Descrição dos campos do superbloco

Área de *bitmap* (gerência dos blocos livres): é a área composta por um ou mais blocos lógicos usados para a gerência de espaço do disco, cujo tamanho está definido em *bitmapSize*. Será utilizado o mecanismo de *bitmap* para registrar a alocação dos blocos do disco: blocos livres e blocos ocupados. AS ROTINAS DE ACESSO A ESSA ÁREA DO DISCO SERÃO FORNECIDAS PELO PROFESSOR.

Área para o diretório raiz: área do disco posicionada logo após os blocos lógicos usados para gerência de espaço do disco. Possui tamanho (em blocos) definido por "*rootSize*". O diretório será organizado em uma estrutura em árvore. Dessa forma, é possível definir subdiretórios. Cada entrada (registro) que identifica um arquivo terá 64 bytes (valor do campo *fileEntrySize*).

Área para os blocos de índice e blocos de dados: nessa área, que se estende até o final do disco, estão alocados os blocos de dados, assim como, em caso de necessidade, os blocos de índices que possuirão ponteiros para blocos de dados dos arquivos.

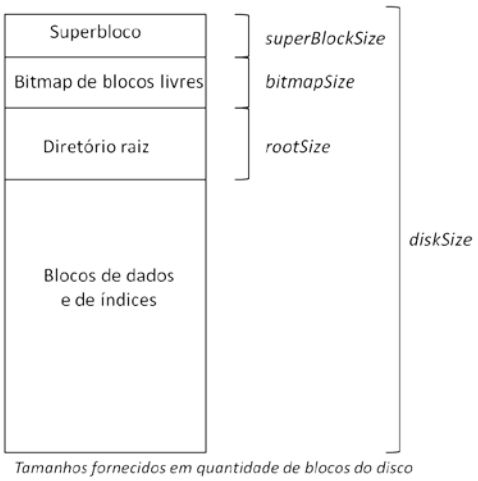

Figura 3 – Organização do disco lógico T2FS

2.1 Implementação de Diretórios no T2FS

O diretório T2FS segue 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 forma absoluta ou relativa. Na forma absoluta, deve-se informar o caminho do arquivo a partir do diretório raiz; na forma relativa, o caminho do arquivo é dado 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 um 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 (regular ou de subdiretório) existente em um disco formatado em T2FS possui uma entrada (registro) em um diretório. Além disso, todos os subdiretórios devem possuir duas entradas: a entrada "." (ponto) e a entrada ".." (ponto ponto), para indicar o próprio diretório (diretório corrente) e o seu diretório pai, respectivamente. **O diretório raiz é exceção, pois não possui diretório pai** . Nesse caso, tanto a entrada "." quanto a entrada ".." devem apontar para o bloco lógico em que o diretório raiz inicia. Os diretórios são implementados através de arquivos organizados internamente como uma lista linear de registros de tamanho fixo.

A tabela 2 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). • 0x00, registro inválido (não associado a nenhum arquivo); • 0x01, arquivo regular; • 0x02, arquivo de diretório. • Outros valores, registro inválido (não associado a nenhum arquivo)
1	32	*name*	Nome do arquivo associado ao registro.
33	7	*Reserved*	Não utilizado
40	4	*blocksFileSize*	Tamanho do arquivo expresso em número de blocos.
44	4	*bytesFileSize*	Tamanho do arquivo expresso em número de bytes.
48	8	*dataPtr[2]*	Dois ponteiros diretos.
56	4	*singleIndPtr*	Ponteiro de indireção simples.
60	4	*doubleIndPtr*	Ponteiro de indireção dupla.

Tabela 2 – Estrutura interna de uma entrada de diretório no T2FS (estrutura *t2fs_record*)

2.2 Implementação de arquivos no T2FS

O T2FS é um sistema de arquivos que emprega o método de alocação indexada e os arquivos podem ser do tipo regular ou diretório. Um arquivo regular é aquele que contém dados de um usuário podendo ser um arquivo texto (caracteres ASC II) ou binário. Sempre que um arquivo for criado, deve ser alocada uma entrada no diretório corrente e os campos da tabela 2 preenchidos de forma adequada.

À medida que um arquivo recebe dados, devem ser alocados os blocos lógicos necessários ao armazenamento desses dados. Sempre que um arquivo tiver sua quantidade de dados reduzida, os blocos lógicos que forem liberados devem ser tornados livres. Ainda, na remoção, a entrada do diretório deve ser atualizada para "inválida" (campo *TypeVal* = 0x00) e todos os blocos lógicos empregados para o arquivo, tanto para dados, como para índices, se for o caso, devem ser marcados como livre. Observe que ao remover uma entrada no diretório NÃO é necessário "compactar" o arquivo de diretório para eliminar essa entrada, basta marcá-la como inválida e reaproveitá-la em uma posterior criação de arquivos.

Ainda, os campos do registro (tabela 2) devem ser atualizados de forma a refletir corretamente o crescimento ou a redução do tamanho de um arquivo.

Os nomes simbólicos para os arquivos e diretórios do T2FS têm no máximo 32 caracteres alfanuméricos. Isso significa que não estão autorizados caracteres especiais, nem espaços em branco, em nomes T2FS e todos os nomes com mais de 32 caracteres devem ser truncados. Não há obrigatoriedade de nenhuma forma de extensão. Os nomes são *case-sensitive*.

Os arquivos são descritos nos diretórios através de um registro chamado *entrada de diretório*. As entradas de diretório têm a mesma forma tanto no diretório raiz como em subdiretórios. O diretório raiz ocupa uma área fixa da partição enquanto que os subdiretórios são implementados como se fossem arquivos. Por ser uma alocação indexada, é necessário fornecer uma lista de quais blocos constituem o arquivo. O T2FS possui essa informação na própria entrada do diretório. Para arquivos com tamanho de até dois blocos lógicos, o campo *dataPrt* informa o endereço desses blocos lógicos. Para arquivos maiores, são empregados, além desses dois ponteiros diretos, os ponteiros de indireção simples ou dupla, também fornecidos na entrada do diretório, que apontam para blocos lógicos que contém a continuação da lista de endereços de blocos lógicos que compõe o arquivo (blocos de índice). O formato dos blocos de índice é descrito na próxima seção.

Como o diretório raiz é uma área fixa no disco T2FS (figura 3), isso significa, que ele terá um número máximo de entradas, limitando assim a quantidade máxima de arquivos (regular e subdiretórios). Assim, não será possível criar mais arquivos que esse limite máximo (a função de criação deverá retornar com erro).

2.3 Formato de um bloco de índices

Os blocos de índices são blocos lógicos do disco onde estão armazenados ponteiros para blocos de dados ou outros blocos de índices, de um determinado arquivo. Cada ponteiro 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 0 (zero), sendo essa a forma de identificar o término da lista de ponteiros.

Dessa forma, levando-se em consideração a estrutura de ponteiros e o disco, o maior tamanho de um arquivo será dado através da seguinte expressão:

$$Tamanho = blockSize.\left(\left(\frac{blockSize}{4}\right)^2 + \left(\frac{blockSize}{4}\right) + 2\right)$$

O final de uma lista de blocos lógicos é sinalizada pelo valor 0 (zero) na entrada.

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 3, onde são usados alguns tipos de dados e protótipos de função que estão definidos no arquivo *t2fs.h* fornecido junto com a especificação deste trabalho. A implementação de seu trabalho deve possuir TODAS AS FUNÇÕES especificadas aqui, mesmo que não tenham sido implementadas. Isso visa evitar erros de compilação com testes que utilizem todas as funções. REFORÇANDO: se você não implementar o corpo de uma função, crie a função conforme o *prototype* fornecido e, em seu corpo, coloque apenas o comando C *return* com um valor apropriado de acordo com o *prototype* da função.

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, DIRENT2 *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 3 – Interface de programação de aplicações – API - da *libt2fs*

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. Os setores lógicos têm, sempre, 256 bytes. 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 Interface de *bitmap* (*bitmap2.o*)

Também como base para a realização do trabalho você receberá o binário *bitmap2.o*, que realiza as operações de alocação e liberação de blocos lógicos da área de dados. Esses blocos são usados para o conteúdo dos arquivos e para os blocos de índice.

As funções dessa API estão descritas a seguir:

`int isBlockFree2 (unsigned int blockNumber)`

Retorna a informação do estado do bloco lógico "`blockNumber`" do disco.

Retorna "0", se o bloco estiver ocupado e um valor diferente de zero, caso esteja livre.

`int freeBlock2 (unsigned int blockNumber)`

Libera o bloco lógico "`blockNumber`". Isso é feito escrevendo na área de bitmap do disco.

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

`int allocBlock2 (unsigned int blockNumber)`

Aloca (ocupa) o bloco lógico "`blockNumber`". Isso é feito escrevendo na área de bitmap do disco.

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

`unsigned int searchFreeBlock2 (void)`

Procura no bitmap de alocação de blocos por um bloco livre.

Retorna o número do bloco encontrado; retorna "0" se não encontrou nenhum bloco livre ou em caso de erro.

Por questões de simplificação, o binário *bitmap2.o*, que implementa as funções de manipulação do *bitmap* de alocação de blocos, e o arquivo de inclusão *bitmap2.h*, com os protótipos dessas funções, serão fornecidos pelo professor.

6 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 os binários fornecidos *apidisk.o* e *bitmap2.o* devem ser ligados usando a seguinte linha de comando:

```
ar crs libt2fs.a <file_1>.o <file_2>.o ... apidisk.o bitmap2.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*".

7 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?

8 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		
	bin	DIRETÓRIO: local onde serão postos os programas executáveis usados para testar a implementação, ou seja, os executáveis dos programas de teste.
	include	DIRETÓRIO: local onde são postos todos os arquivo ".h". Nesse diretório deve estar o "*t2fs.h*", o "*apidisk.h*" e o "*bitmap2.h*"
	lib	DIRETÓRIO: local onde será gerada a biblioteca "*libt2fs.a*". (junção da "*t2fs*" com "*apidisk.o*" e "*bitmap2.o*"). Os binários *apidisk.o* e *bitmap2.o* também serão postos neste diretório.
	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.
	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.

9 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.

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

1. Faz parte da avaliação a obediência RÍGIDA aos padrões de entrega definidos na seção 8 (arquivos *tar.gz*, *makefiles*, estruturas de diretórios, etc);
2. O trabalho pode ser feito em grupos de até DOIS alunos (grupos com mais de dois 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 **21 de JUNHO de 2016**;
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.

11 Observações

Recomenda-se a troca de ideias entre os alunos. Entretanto, a identificação de cópias de trabalhos acarretará na aplicação do Código Disciplinar 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.