Trabalho Prático II – ENTREGA: 11 de Julho de 2017

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 uma variante de alocação indexada encadeada para a criação de arquivos e diretórios.

Esse Sistema de Arquivos será chamado, daqui para diante, de T2FS (*Task 2 – File System – Versão 2017.1*) 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 (ver anexo C). As funções básicas de leitura e escrita simulam as solicitações enviadas ao *driver* de dispositivo (disco T2FS).

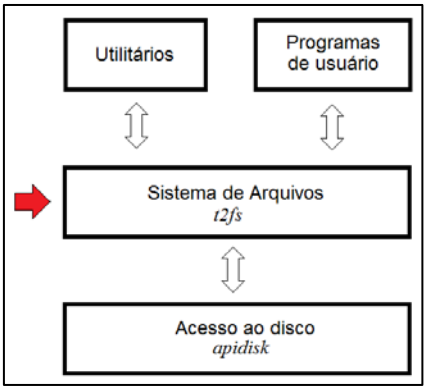

Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

2 Estrutura de um volume T2FS

O espaço disponível no disco formatado logicamente para T2FS (volume) está dividido em três áreas: bloco de *boot*, *Master File Table* (MFT) e blocos de dados. No bloco de boot está descrita a cartografia do volume, na área da MFT estão os registos de descritores dos arquivos de dados ou de diretório, e a área de blocos de dados contém os dados dos arquivos e as entradas de diretórios.

Um volume T2FS é organizado em blocos, e estes, por sua vez, são organizados em uma sequência contígua de setores. Um volume será formado por *B* blocos, numerados de 0 a *B-1*, e um bloco será formado por *N* setores contíguos Dessa forma, um volume será formado por *S* setores, numerados de 0 a *S-1*, onde *S = B x N*.

Nessa organização, o primeiro setor do bloco de boot (bloco 0) corresponde ao setor 0 do volume. A cartografia do volume T2FS está descrita nesse bloco, conforme mostra a Figura 2 e detalhada na Tabela 1. Um setor possui 256 bytes.

Tabela 1 – Descrição dos campos do bloco de *boot*

Posição relativa	Tamanho (bytes)	Nome	Valor	Descrição
0	4	*id*	"T2FS"	Identificação do sistema de arquivo. É formado pelas letras "T2FS".
4	2	*version*	0x7F01	Versão atual desse sistema de arquivos Valor fixo 0x7F0=2017; 1=1º semestre.
6	2	*blockSize*	0x0004	Quantidade de setores que formam um bloco lógico
8	2	*MFTBlocksSize*	0x0800	Quantidade de blocos (conjunto de *n* setores) usados para armazenar a *Master File Table*
16	4	*diskSectorSize*	0x8000	Quantidade total de setores na partição T2FS. Inclui o bloco de *boot*, MFT e blocos de dados
20 até o final (*blocksize* - 1)		*reservado*		Não usados

Bloco de *boot*: é a área que descreve a cartografia do disco. Essa área **inicia no primeiro setor do disco**, conforme aparece na tabela 1. Todos os valores no bloco de *boot* são armazenados em um formato *little-endian* (a parte menos significativa do valor é armazenada no endereço mais baixo de memória).

Área de MFT: conjunto de blocos do disco onde estão armazenados os registros de descritores (índices) de arquivos do T2FS. O tamanho dessa área, em número de blocos, é dado por *MFTBlockSize*, fornecido no bloco de *boot*. Os quatro primeiros registros MFT são reservados para o uso do sistema de arquivos, conforme a seguir:

- Registro 0 (zero): descritor do arquivo que fornece o *bitmap* de blocos de dados livres e ocupados;
- Registro 1 (um): descritor do arquivo associado ao diretório raiz;
- Registros 2 e 3: descritores reservados para uso futuro.

Os registros restantes do MFT (registro 4 em diante) serão usados para armazenar os descritores dos arquivos de dados e de diretórios. Esses descritores contêm os ponteiros de blocos de dados que formam o arquivo.

Área de blocos de dados: área que inicia no primeiro bloco após a área de MFT e se estende até o final do disco. Nessa área estão os blocos de dados que formam os arquivos e eventuais blocos de índices, que serão usados quando for esgotada a capacidade do registro no MFT.

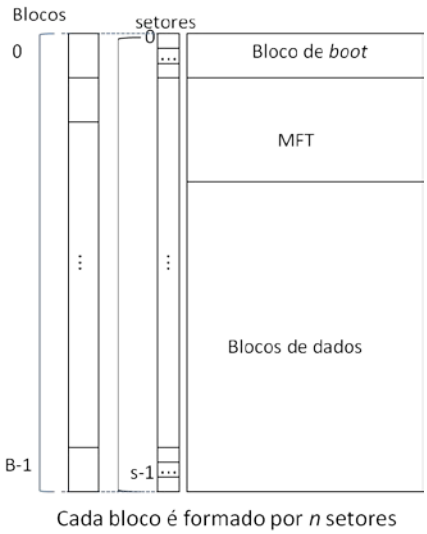

Figura 2 – Organização do disco lógico T2FS

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 nomes dos arquivos no T2FS são dados apenas na forma absoluta, ou seja, o caminho do arquivo é sempre fornecido a partir do diretório raiz. O caractere de barra ("/") será utilizado na formação desses caminhos absolutos. Na criação de um arquivo, todos os diretórios intermediários da raiz até ao diretório corrente já devem existir. Se não existirem, a primitiva de criação deverá retornar com erro. Por exemplo, ao criar o *arqx* com o caminho */a/b/c/d/arqx* todos os diretórios do caminho já devem existir (*a, b, c* e *d*).

Cada arquivo (regular ou subdiretório) existente em um disco formatado em T2FS possui uma entrada (registro) em um diretório. Os diretórios são implementados por 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 são armazenados em formato *little-endian*.

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

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	51	*name*	Nome do arquivo associado ao registro.
52	4	*blocksFileSize*	Tamanho do arquivo expresso em número de blocos.
56	4	*bytesFileSize*	Tamanho do arquivo expresso em número de bytes.
60	4	*MFTNumber*	Número do registro MFT

2.2 Implementação de arquivos no T2FS

Os arquivos no T2FS podem ser do tipo regular ou diretório. Um arquivo regular é aquele que contém dados de um usuário, podendo conter texto (caracteres ASC II) ou valores binário. Sempre que um arquivo for criado, deve-se alocar uma nova entrada no diretório corrente (conforme formato da Tabela 2) e preenchidos adequadamente os seus campos.

À medida que um arquivo recebe dados, devem ser alocados os blocos lógicos necessários ao seu armazenamento. Sempre que um arquivo tiver sua quantidade de dados reduzida, os blocos lógicos que forem liberados devem ser tornados livres.

No caso da remoção de um arquivo, além da liberação dos blocos lógicos de dados, a entrada do diretório deve ser atualizada para "inválida" (campo *TypeVal* = 0x00). 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.

Os campos da entrada de diretório (estrutura *t2fs_record* detalhada na tabela 2) devem ser atualizados de forma a refletir corretamente o crescimento ou a redução do tamanho do arquivo.

O tamanho máximo dos nomes simbólicos dos arquivos e diretórios do T2FS está definido pelo tamanho do campo *name* da estrutura *t2fs_record*. Os caracteres usados nos nomes simbólicos podem ser letras, números ou o caractere "." (ponto). Os nomes simbólicos não necessitam ter extensão e são *case-sensitive*.

O T2FS é um sistema de arquivos que emprega uma variante do método de alocação indexada encadeada. Portanto, é necessário fornecer uma lista dos blocos que constituem o arquivo. No T2FS os blocos do disco são identificados pelo seu *logical block number* (LBN), que inicia em 0 (zero) e vai até *B-1*. De forma semelhante, os blocos lógicos componentes dos arquivos são identificados pelo *virtual block number* (VBN), e iniciam em 0 (zero) e estendem-se até *i*-1, onde *i* é a quantidade de blocos que formam o arquivo. Esses VBNs devem ser mapeados para os LBNs.

O atributo *MFTNumber*, existente na entrada do diretório T2FS (estrutura *t2fs_record*)), fornece o número do registro do MFT onde se encontra o mapeamento entre VBNs e LBNs.

Os registros do MFT com as informações de mapeamento entre VBNs e LBNs têm um tamanho fixo de 512 bytes composto por trinta e duas 4-tuplas (Figura 3). A Tabela 3 apresenta o formato de uma 4-tupla dado pela estrutura *t2fs_MFT*. Novamente, todos os valores estão em um formato *little-endian*.

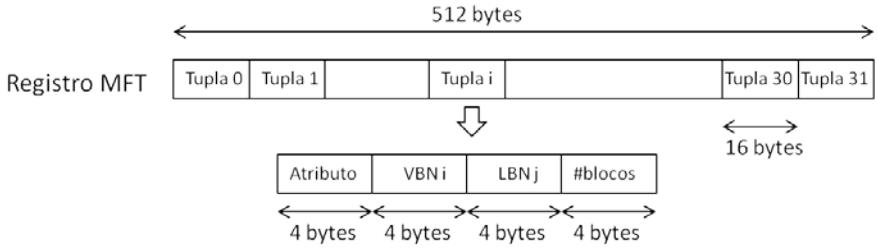

Figura 3 – Registro MFT e suas 4-tuplas

Tabela 3 – Formato de uma 4-tupla do MFT (estrutura *t2fs_MFT*)

Posição relativa	Tamanho (bytes)	Nome	Descrição
0	4	*atributeType*	-1: registro MFT livre 0: marcador de fim de encadeamento 1: tupla de mapeamento VBN-LBN 2: registro MFT adicional
4	4	*virtualBlockNumber*	VBN inicial *k* ou registro MFT adicional
8	4	*logicalBlockNumber*	LBN inicial *j*
12	4	*numberOfContiguosBlocks*	Quantidade de blocos contíguos ocupados por essa porção do arquivo

A figura 4 fornece um exemplo do emprego da alocação indexada encadeada. Nesse exemplo, o arquivo "texto.doc" é composto por sete blocos virtuais numerados de 0 a 6. No disco, esses blocos virtuais (VBN) foram alocados para os blocos lógicos (LBN) que correspondem a três regiões: 3 blocos contíguos a partir do LBN 56 (região 1: LBNs 56, 57 e 58), 2 blocos contíguos iniciando no LBN 80 (região 2: LBNs 80 e 81) e mais 2 blocos contíguos começando no LBN 99 (região 3: blocos 99 e 100).

Um ponto importante a ressaltar é que SEMPRE se deve tentar manter o mapeamento de forma mais contígua possível. Assim, esse exemplo pressupõe um disco que já estava fragmentado e foram encontrados, de acordo com uma política FIRST-FIT, essas três regiões livres. Note que a melhor situação seria obter 7 LBNs contíguos. Nesse caso, no registro MFT, haveria apenas a tupla (0, 523, 7) para descrever o mapeamento, considerando a existência de 7 LBNs livres a partir do LBN 523.

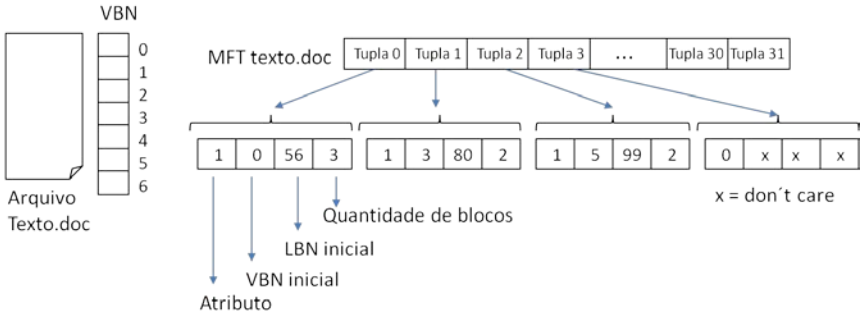

Figura 4 – Exemplo de mapeamento de VBNs a LBNs e de um registro MFT

Caso um arquivo necessite mais 4-tuplas do que aquelas disponíveis em um registro de MFT, a última 4-tupla (trigésima primeira 4-tupla) do registro não conterá informação de mapeamento. No seu lugar, essa última 4-tupla deverá conter o número do registro de MFT onde a sequência de mapeamento deve continuar. Essa última 4-tupla deverá receber o *attributeType* = 2 (registro MFT adicional) e o número do próximo registro MFT no campo *virtualBlockNumber*.

As informações de blocos livres e ocupados serão armazenadas no volume usando a técnica de *bitmap*. Para isso, o arquivo de nome $BitMap, cujo descritor está no registro 0 do MFT, contém as informações dos blocos lógicos livres e ocupados.

O arquivo com as entradas do diretório raiz está descrito no registro 1 (um) do MFT.

3 Interface de Programação da T2FS (libt2fs.a)

Sua tarefa é implementar a biblioteca *libt2fs.a* que possibilitará o acesso aos arquivos regulares e de diretório do sistema de arquivos T2FS.

As funções a serem implementadas estão resumidas na tabela 4, 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 alguma das funções, 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. Sugere-se um valor que indique erro de execução.

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

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

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 truncate2 (FILE2 handle)`	Função usada para truncar um arquivo. Remove do arquivo todos os bytes a partir da posição atual do contador de posição (*current pointer*), inclusive, até o seu final.
`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 diretório.

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.

Importante: a biblioteca *apidisk* considera que o arquivo que emula o disco virtual T2FS possui sempre o nome *t2fs_disk.dat* e, esse arquivo deve estar localizado no mesmo diretório em que estão os programas executáveis que o utiliza.

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 do disco.

Tabela 5 – Interface das funções de bitmap

Nome	Descrição
`int getBitmap2 (` **`int blockNumber)`**	Retorna a informação de alocação do bloco "***blockNumber***", da área de bitmap. A função retorna os seguintes valores: • se retornar "1" (um), o bloco está alocado; • se retornar "0" (zero), o bloco está livre; • se retornar um valor negativo, houve erro.
`int setBitmap2 (` **`int blockNumber,`** **`int allocated)`**	Seta a informação de alocação do bloco "***blockNumber***". Essa informação é passada no parâmetro "***allocated***". Esse parâmetro pode ser: • "0" (zero), se o bloco deve ser desalocado (liberado); • outros valores, se o bloco deve ser alocado. A função retorna os seguintes valores: • se retornar "0" (zero), operação realizada com sucesso; • se retornar um valor negativo, houve erro.
`int searchBitmap2 (` **`int allocated)`**	Procura, a partir do bloco "0" (zero), na área de bitmap, um bloco no estado indicado por "***allocated***", que pode ser: • "0" (zero), se deve procurar por um bloco desalocado (livre); • outros valores, se deve procurar por um bloco alocado. A função retorna os seguintes valores: • se retornar "0" (zero), não foi encontrado bloco no estado solicitado. Observar que o bloco "0" é o bloco de boot. Dessa forma, sempre alocado; • se retornar um valor positivo (diferente de zero), foi encontrado um bloco no estado solicitado e retornado o seu número; • se retornar um valor negativo, houve 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 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*"

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.

7 Avaliação

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

- Entrega do trabalho final dentro dos prazos estabelecidos;
- Obediência à especificação (formato e nome das funções);
- Compilação e geração da biblioteca sem erros ou warnings;
- Fornecimento de todos os arquivos solicitados conforme organização de diretórios fornecidos na seção 6;
- Todas as funções devem ter "contrato". Deve constar no contrato o(s) componente(s) do grupo responsável(is) pela implementação;
- Execução correta dentro da máquina virtual alunovm-sisop.ova.

Uma vez cumpridas as condições gerais, as implementações serão avaliadas e valorizadas da seguinte forma:

- 50,0 pontos de avaliação individual, referentes às **reuniões de acompanhamento**;
- 50,0 pontos de avaliação da implementação, referentes ao **funcionamento da biblioteca**.

Reuniões de acompanhamento: serão observados os seguintes itens:

- a correta associação entre a implementação e os conceitos vistos em aula;
- a gestão do tempo de desenvolvimento;
- distribuição das atividades entre os componentes do grupo;
- planejamento e projeto inicial;
- evolução da implementação conforme o projeto, e realização de ajustes quando necessário.

Funcionamento da biblioteca: será realizada pela observação dos seguintes itens:

- utilização de programas padronizados desenvolvidos pelo professor, para uso da biblioteca;
- clareza e organização do código;
- o uso de programação modular;
- correta utilização dos *makefiles*;

- correta utilização dos arquivos de header (sem código "C" dentro dos includes).

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

- Faz parte da avaliação a obediência RÍGIDA aos padrões de entrega definidos na seção 6 (arquivos *tar.gz*, estrutura de diretórios, *makefile*, *etc*);
- O trabalho deverá ser desenvolvido em grupos com três componentes;
- As reuniões de projeto serão avaliadas e as notas alcançadas farão parte da nota final do trabalho. A ausência nas reuniões de projeto implicam em nota 0 (zero) naquela avaliação;
- O trabalho deverá ser apresentado no laboratório, no horário da aula, na data de **11 de julho de 2017**;
- Para fazer a apresentação do trabalho no laboratório, um dos componentes do grupo deve realizar o *upload* de um arquivo ***tar.gz*** com suas implementações até a data estabelecida;
- Após a apresentação no laboratório os grupos poderão realizar o *upload* de um segundo arquivo *tar.gz* com eventuais ajustes de suas implementações realizados durante a apresentação;
- Não serão aceitos trabalhos entregues além dos prazos estabelecidos;
- 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.

ANEXO A – Compilação e Ligação

1. Compilação de arquivo fonte para arquivo objeto

Para compilar um arquivo fonte (*arquivo.c*, por exemplo) e gerar um arquivo objeto (*arquivo.o*, por exemplo), pode-se usar a seguinte linha de comando:

```
gcc –c arquivo.c –Wall
```

Notar que a opção –*Wall* solicita ao compilador que apresente todas as mensagens de alerta (*warnings*) sobre possíveis erros de atribuição de valores a variáveis e incompatibilidade na quantidade ou no tipo de argumentos em chamadas de função.

2. Compilação de arquivo fonte DIRETAMENTE para arquivo executável

A compilação pode ser feita de maneira a gerar, diretamente, o código executável, sem gerar o código objeto correspondente. Para isso, pode-se usar a seguinte linha de comando:

```
gcc –o arquivo arquivo.c –Wall
```

3. Geração de uma biblioteca estática

Para gerar um arquivo de biblioteca estática do tipo "*.a*", os arquivos fonte devem ser compilados, gerando-se arquivos objeto. Então, esses arquivos objeto serão agrupados na biblioteca. Por exemplo, para agrupar os arquivos "*arq1.o*" e "*arq2.o*", obtidos através de compilação, pode-se usar a seguinte linha de comando:

```
ar crs libexemplo.a arq1.o arq2.o
```

Nesse exemplo está sendo gerada uma biblioteca de nome "*exemplo*", que estará no arquivo *libexemplo.a*.

4. Utilização de uma biblioteca

Deseja-se utilizar uma biblioteca estática (chamar funções que compõem essa biblioteca) implementada no arquivo *libexemplo.a*. Essa biblioteca será usada por um programa de nome *myprog.c*.

Se a biblioteca estiver no mesmo diretório do programa, pode-se usar o seguinte comando:

```
gcc -o myprog myprog.c –lexemplo -Wall
```

Notar que, no exemplo, o programa foi compilado e ligado à biblioteca em um único passo, gerando um arquivo executável (arquivo *myprog*). Observar, ainda, que a opção –*l* indica o nome da biblioteca a ser ligada. Observe que o prefixo *lib* e o sufixo *.a* do arquivo não necessitam ser informados. Por isso, a menção apenas ao nome *exemplo*.

Caso a biblioteca esteja em um diretório diferente do programa, deve-se informar o caminho (*path* relativo ou absoluto) da biblioteca. Por exemplo, se a biblioteca está no diretório */user/lib*, caminho absoluto, pode-se usar o seguinte comando:

```
gcc -o myprog myprog.c –L/user/lib –lexemplo -Wall
```

A opção "*-L*" suporta caminhos relativos. Por exemplo, supondo que existam dois diretórios: *testes* e *lib*, que são subdiretórios do mesmo diretório pai. Então, caso a compilação esteja sendo realizada no diretório *testes* e a biblioteca desejada estiver no sudiretório *lib*, pode-se usar a opção *-L* com "*../lib*". Usando o exemplo anterior com essa nova localização das bibliotecas, o comando ficaria da seguinte forma:

```
gcc -o myprog myprog.c –L../lib –lexemplo -Wall
```

ANEXO B – Compilação e Ligação

1. Desmembramento e descompactação de arquivo *.tar.gz*

O arquivo *.tar.gz* pode ser desmembrado e descompactado de maneira a gerar, em seu disco, a mesma estrutura de diretórios original dos arquivos que o compõe. Supondo que o arquivo *tar.gz* chame-se "*file.tar.gz*", deve ser utilizado o seguinte comando:

```
tar -zxvf file.tar.gz
```

2. Geração de arquivo *.tar.gz*

Uma estrutura de diretórios existente no disco pode ser completamente copiada e compactada para um arquivo *tar.gz*. Supondo que se deseja copiar o conteúdo do diretório de nome "*dir*", incluindo seus arquivos e subdiretórios, para um único arquivo *tar.gz* de nome "*file.tar.gz*", deve-se, a partir do diretório pai do diretório "*dir*", usar o seguinte comando:

```
tar -zcvf file.tar.gz dir
```

ANEXO C – Discos físicos

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 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 lógico T2FS é formado por uma sequência contígua de *n* setores lógicos. A figura C.1 ilustra esses conceitos de forma genérica.

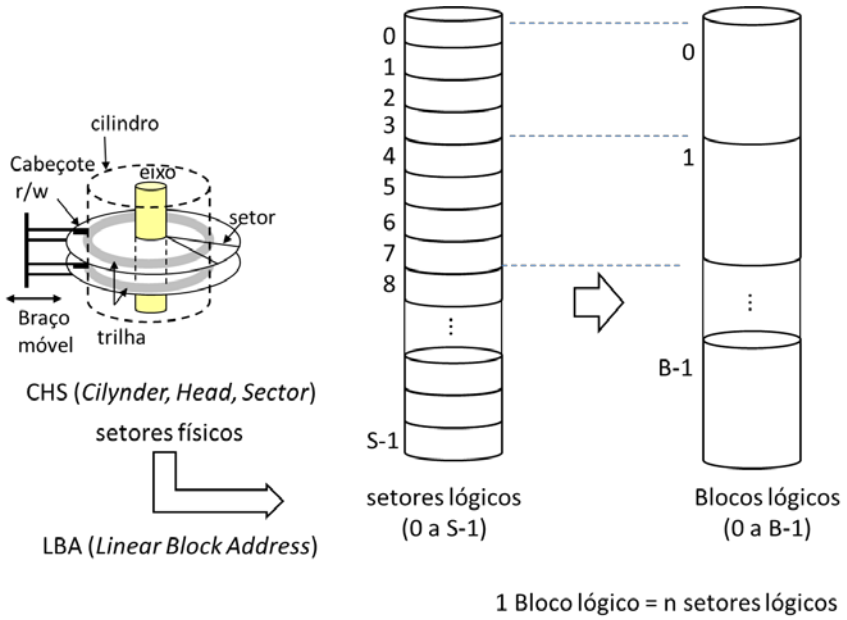

Figura C.1 – setores físicos, setores lógicos e blocos lógicos (diagrama genérico)