Trabalho Prático II – ENTREGA: 12 de Junho de 2018

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 combinada para a criação de arquivos e diretórios.

Esse sistema de arquivos será chamado, daqui para diante, de T2FS (*Task 2 – File System – Versão 2018.1*) 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 um disco formatado com esse 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).

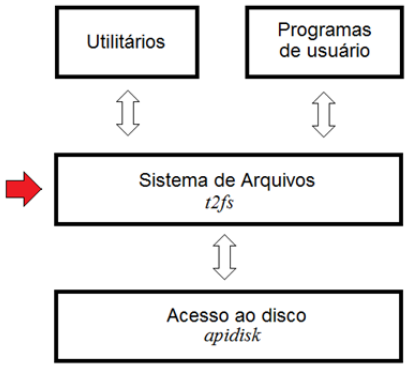

Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

2 Estrutura de um volume T2FS

O espaço disponível no disco formatado logicamente para T2FS (volume) está dividido em cinco áreas: superbloco, *bitmap* de blocos do disco livres e ocupados, *bitmap* de *i-nodes* livres e ocupados, *i-nodes* e, por fim, blocos de dados. A gerência do espaço em disco é feito por alocação indexada combinada e o diretório segue uma hierarquia em árvore. Um volume T2FS é formado por *S* setores (0 a *S-1*), onde os primeiros setores armazenam as estruturas gerenciais que definem e controlam a cartografia do disco T2FS, e os setores restantes são agrupados, *n* a *n*, formando um bloco de dados. Portanto, os blocos T2FS são formados por *n* setores contíguos e são numerados de 0 a *B-1*, onde *B* é o número total de blocos no disco. Esses valores são fornecidos no superbloco (vide tabela 1) e a figura 2 mostra a estrutura do volume T2FS. As áreas do disco T2FS são detalhadas na sequência.

Tabela 1 – Descrição dos campos do superbloco

Posição relativa	Tamanho (bytes)	Nome	Valor	Descrição
0	4	*id*	"T2FS"	Identificação do sistema de arquivo. É formado pelas letras "T2FS".
4	2	*version*	0x7E21	Versão atual do sistema de arquivos: (valor fixo 0x7E2=2018; 1=1º semestre).
6	2	*superblockSize*	1	O superbloco ocupa o primeiro bloco do disco, isso é, o bloco zero.
8	2	*freeBlocksBitmapSize*	b_b	Quantidade de blocos usados para armazenar o *bitmap* de blocos de dados livres e ocupados.
10	2	*freeInodeBitmapSize*	b_i	Quantidade de blocos usados para armazenar o *bitmap* de *i-nodes* livres e ocupados.
12	2	*inodeAreaSize*	i	Quantidade de blocos usados para armazenar os *i-nodes* do sistema.
14	2	*blockSize*	n	Quantidade de setores que formam um bloco lógico.
16	4	*diskSize*	d	Quantidade total de blocos na partição T2FS. Inclui o superbloco, áreas de *bitmap*, área de *i-node* e blocos de dados
20 até o final		*reservado*		Não usados

Superbloco: é a área de controle do sistema de arquivos. Essa área **ocupa o primeiro bloco do disco**, conforme aparece na tabela 1. Todos os valores no superbloco são armazenados em um formato *little-endian* (a parte menos significativa do valor é armazenada no endereço mais baixo de memória). Observe que os valores da tabela 1 correspondem ao primeiro setor do disco.

Área de *bitmap* de blocos livres/ocupados: é formada pelo conjunto de b_b blocos do disco, onde b_b é o valor definido em *freeBlocksBitmapSize* fornecido no superbloco. Cada *bit* nessa área corresponde a um bloco no disco e indica se o bloco está livre ou ocupado. (OBS.: As funções para alocar e liberar blocos via *bitmap* são fornecidas como parte integrante desta especificação. É obrigatório o uso dessas funções na implementação do T2FS). Se esse bit vale "1", o bloco está ocupado; se vale "0", está livre.

Área de *bitmap* de *i-nodes* livres/ocupados: é formada pelo conjunto de b_i blocos do disco, onde b_i é o valor definido em *freeInodeBitmapSize* fornecido no superbloco. Cada *bit* nessa área corresponde a um *i-node* no disco e indica se o *i-node* está livre ou ocupado. (OBS.: As funções para alocar e liberar *i-nodes* via *bitmap* são fornecidas como parte integrante desta especificação. É obrigatório o uso dessas funções na implementação do T2FS). Se esse bit vale "1", o *i-node* está ocupado; se vale "0", está livre.

Área de *i-nodes*: conjunto de blocos do disco onde estão armazenados os *i-nodes* do T2FS. O tamanho dessa área, em número de blocos, é dado por *inodeAreaSize*, fornecido no superbloco.

Área para blocos de dados: área que inicia no primeiro bloco após a área de *i-nodes* e se estende até o final do disco. É nessa área que estão os blocos de dados que formarão os arquivos de dados, arquivos de diretórios e blocos de índices. Se necessário, em função do tamanho de um arquivo, esses blocos poderão ser empregados como blocos de índices, ou seja, possuirão ponteiros para blocos de dados dos arquivos.

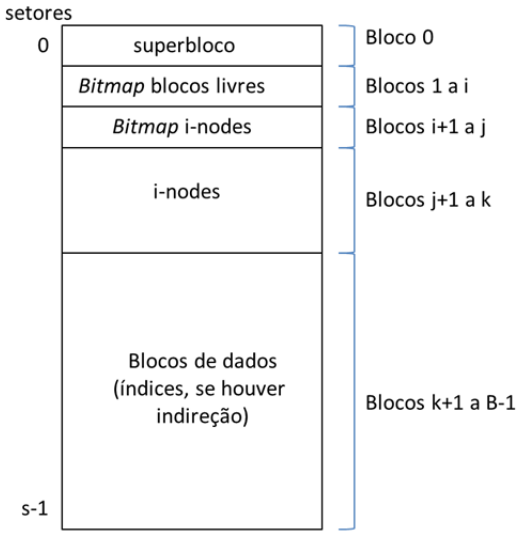

Figura 2 – Organização do disco lógico T2FS

2.1 Hierarquia de arquivos no T2FS

O T2FS é um sistema de arquivos que emprega diretório em árvore e utiliza o método de alocação indexada para organização de arquivos e gerenciamento de espaço do disco. Por possuir uma hierarquia em árvore, os arquivos T2FS podem ser especificados através de seus caminhos de forma absoluta ou relativa (*pathname*).

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 um caminho absoluto, caso contrário, como caminho relativo. A notação a ser empregada para descrever os caminhos relativos é a mesma do Unix, isso é, a combinação de caracteres "./" indica o diretório corrente, e a sequência de caracteres "../" indica o diretório-pai. Assim, um arquivo, no diretório corrente, pode ser acessado de duas formas: (i) fazendo referência apenas ao seu nome; ou (ii) precedendo o nome pelo conjunto de caracteres "./" para indicar que é relativo ao atual diretório corrente.

2.2 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:

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

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	59	*name*	Nome simbólico do arquivo.
60 a 63	4	*inodeNumber*	Número do *i-node*

As dois primeiros registros de um arquivo de diretório correspondem, respectivamente, aos registros que apontam para o *i-node* do próprio diretório (entrada ".") e diretório pai (entrada ".."). Todo arquivo de diretório deve possuir esses dois registros.

O diretório raiz, por ser o primeiro diretório da hierarquia, apresenta duas exceções. Primeira, a localização do primeiro bloco do arquivo do diretório raiz é dado pelo *i-node* zero. Segunda, a entrada "..", a que fornece o registro do diretório pai, deve apontar para ele mesmo (diretório corrente), já que o diretório raiz não tem diretório pai.

2.3 Implementação de arquivos no T2FS

Os arquivos T2FS 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 arquivo de diretório onde o arquivo será inserido e os campos dessa entrada (registro) preenchidos de forma adequada conforme a tabela 2.

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.

Os arquivos são descritos nos diretórios através de um registro chamado *entrada de diretório* (*estrutura t2fs_record*). Os nomes simbólicos para os arquivos e diretórios do T2FS têm no máximo 58 caracteres alfanuméricos (0-9, a-z e A-Z). Os nomes simbólicos não necessitam ter extensão e são *case-sensitive*. Os nomes simbólicos deverão ser implementados como *strings* em C.

No caso da remoção de um arquivo, a entrada do diretório deve ser atualizada para "inválida" (campo *TypeVal* = TYPEVAL_INVALIDO, conforme definido no arquivo de inclusão *t2fs.h* e todos os blocos lógicos empregados para o arquivo, tanto para dados, quanto para índices, se for o caso, devem ser marcados como livres. 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.

Por empregar alocação indexada combinada, é necessário fornecer uma lista de quais blocos constituem o arquivo. O T2FS possui essa informação no *i-node* fornecido na entrada do diretório através do atributo *inodeNumber*. O formato do *i-node* T2FS é dado na tabela 3 e forma a estrutura *t2fs_inode*. Novamente, todos os valores estão em um formato *little-endian*.

Assim, para arquivos com tamanho de até dois blocos lógicos, o campo *dataPrt* informa o endereço desses blocos lógicos. Para arquivos maiores que dois blocos lógicos, são empregados, além desses dois ponteiros diretos, os ponteiros de indireção simples, ou dupla, os quais apontam para blocos lógicos que contém a continuação da lista de endereços de blocos lógicos que compõe o arquivo (bloco de índice). Os ponteiros não utilizados devem ser preenchidos com o valor de ponteiro inválido (INVALID_PTR), conforme definido no arquivo de inclusão *t2fs.h*. Os campos do *i-node* que informam o tamanho do arquivo devem ser atualizados de forma a refletir corretamente o crescimento ou a redução de um arquivo.

Tabela 3 – Estrutura interna de um *i-node* T2FS (estrutura *t2fs_inode*)

Posição relativa	Tamanho (bytes)	Nome	Descrição
0	4	*blocksFileSize*	Tamanho do arquivo expresso em quantidade de blocos
4	4	*bytesFileSize*	Tamanho do arquivo expresso em bytes
8	8	*dataPtr[2]*	Dois ponteiros diretos.
16	4	*singleIndPtr*	Ponteiro de indireção simples.
20	4	*doubleIndPtr*	Ponteiro de indireção dupla.
24	8	reservado	Não usados

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 definidos no arquivo *t2fs.h,* fornecido junto com a especificação deste trabalho. A implementação de seu trabalho deve possuir TODAS AS FUNÇÕES aqui especificadas, 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 10 (dez) arquivos regulares abertos simultaneamente. Notar que é possível abrir o mesmo arquivo mais de uma vez.

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*) em 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 para *offset*.
`int mkdir2 (char *pathname)`	Função usada para criar um novo diretório.
`int rmdir2 (char *pathname)`	Função usada para remover (apagar) um diretório do disco.
`int chdir2(char *pathname)`	Função usada para alterar o CP (*current path*)
`int getcwd2 (char *pathname, int size)`	Função usada para obter o caminho do diretório corrente.
`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 são endereçados através de sua numeração sequencial a partir de zero. Os setores lógicos têm, sempre, 256 bytes. As funções dessa API são:

`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 que emulará o disco onde estará o sistema de arquivos T2FS (arquivos *.dat*).

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 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 todos os binários resultantes da compilação dos programas fonte C existentes no diretório src.
	exemplo	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 "*bitmap.o*"). Os binário *apidisk.o* e *bitmap.o* também serão armazenados 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.
	t2fs_disk.dat	ARQUIVO: arquivo que emula o disco T2FS

6 Avaliação

A avaliação do trabalho considerará as seguintes condições:

- Entrega dos formulários de acompanhamento (parciais e final) e 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 dada na seção 5;
- Execução correta dentro da máquina virtual *alunovm-sisop.ova.*

Itens que serão avaliados e sua valoração:

- 20,0 pontos: relativos ao preenchimento adequado dos formulários de acompanhamento e de entrega final disponibilizados no Moodle. Cada formulário vale 4 pontos: são quatro relatórios de acompanhamento (parciais) e um de entrega final. ATENÇÃO: respeite os prazos do Moodle. Não serão aceitos preenchimento após o prazo.
- 80,0 pontos: funcionamento da biblioteca de acordo com a especificação. Para essa verificação serão utilizados programas padronizados desenvolvidos pelos professores da disciplina. A nota será proporcional à quantidade de execuções corretas desses programas, considerando-se a dificuldade relativa de cada um.

7 Avisos Gerais – LEIA com MUITA ATENÇÃO!!!

1. O trabalho deverá ser desenvolvido em grupos de TRÊS componentes.
2. Os relatórios de acompanhamento parciais e final serão avaliados e as notas alcançadas farão parte da nota final do trabalho. A não entrega de um relatório (parcial ou final) implica em nota 0 (zero) na sua avaliação. As entregas deverão ser feitas até a data prevista, conforme cronograma de entrega no *Moodle*. NÃO haverá extensão de prazos.
3. O trabalho final deverá ser entregue até a data prevista. Deverá ser entregue um arquivo *tar.gz* conforme descrito na seção 5. NÃO haverá extensão de prazos.

8 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.

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. Note 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 empregando, como exemplo, que um bloco lógico é formado por quatro setores lógicos.

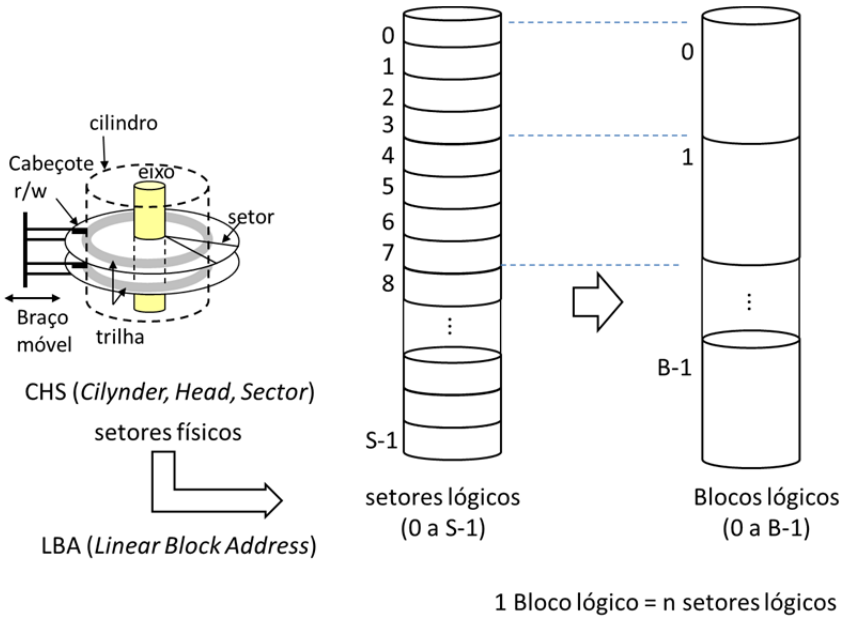

Figura C.1 – setores físicos, setores lógicos e blocos lógicos (diagrama genérico)