Trabalho Prático II – ENTREGA: 14 de novembro de 2019

Implementação de um Sistema de Arquivos T2FS (revisão 21.10.19)

1 Descrição Geral

O objetivo deste trabalho é a aplicação dos conceitos de sistemas operacionais na implementação de um sistema de arquivos que será chamado, daqui para diante, de T2FS (*Task 2 – File System – Versão 2019.2*) 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 nesse 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. O programa de testes padrão é um *shell* (*shell2*) fornecido junto com os arquivos deste trabalho.

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 com pelo menos uma partição formatada em T2FS, e por funções básicas de leitura e escrita de **setores** desse disco. As funções básicas de leitura e escrita simulam as solicitações enviadas ao *driver* de dispositivo (disco T2FS).

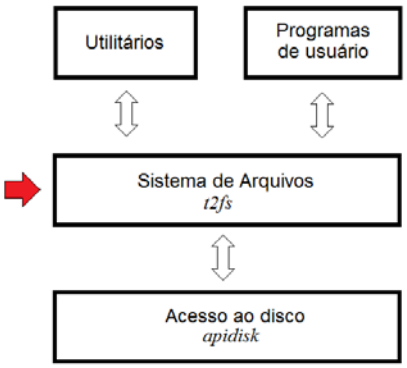


Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

2 O sistema de Arquivos T2FS

O sistema de arquivos T2FS emprega uma estrutura linear (único nível) para diretórios e pode armazenar arquivos regulares (binário ou texto) de tamanhos que podem variar desde zero bytes até a quantidade de blocos disponíveis no disco.

Por seguir uma estrutura de diretório em um único nível, o T2FS possui apenas um arquivo do tipo diretório, o diretório raiz. O diretório raiz, em função da quantidade de arquivos criados, terá um número variável de entradas. Lembrando que cada entrada no diretório corresponde a um registro que mantém os metadados de um arquivo. Por ser uma hierarquia em um único nível não existem caminhos relativos e absolutos, ou seja, todos os arquivos são referenciados a partir do diretório raiz. O registro de uma entrada T2FS é dado na Tabela 1.

Tabela 1 – 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 *link*). • 0x00, registro inválido (não associado a nenhum arquivo); • 0x01, arquivo regular (binário ou texto); • 0x02, *link* simbólico
1	51	*name*	Nome do arquivo: *string* com caracteres ASCII (0x21 até 0x7A), *case sensitive*. O espaço não ocupado nesta entrada deve ser preenchido com o caractere especial "\0" (0x00). O último caractere dessa área deve ter, sempre, um "\0".
52	8	*Não usado*	Reservado para uso futuro
60	4	*inodeNumber*	Identificador do *i-node* associado ao arquivo

Os tipos de arquivos suportados pelo T2FS são dois: arquivo (binário ou texto) e vínculos simbólicos (*softlinks*). O T2FS também oferece suporte a vínculos estritos (*hardlinks*). Cada arquivo (e *link*) do T2FS é identificado por um nome simbólico formado por até 50 caracteres alfanuméricos (0-9, a-z e A-Z). Os nomes são *case-sensitive*.

Um *softlink* T2FS é implementado através de um arquivo com apenas UM bloco, cujo conteúdo é um *string* com o nome do arquivo para qual o *softlink* faz referência. Lembre-se que os *strings* em C possuem o caracter '\0' como terminador. Já os *hardlinks* T2FS são implementados apontando diretamente para a estrutura de dados associada ao arquivo, ou seja, seu *i-node* T2FS.

O gerenciamento do espaço em disco aborda a forma como é feita a alocação dos blocos de disco para os arquivos (na criação e no crescimento) e o controle dos blocos de disco livres e ocupados. O T2FS emprega um esquema de alocação indexada combinada (semelhante aos *i-nodes* Unix) e *bitmaps* para indicar quais *i-nodes* e blocos de disco estão livres ou ocupados. A tabela 2 fornece os campos que formam um *i-node* T2FS.

IMPORTANTE: o arquivo que representa o diretório raiz está associado ao *i-node* T2FS zero.

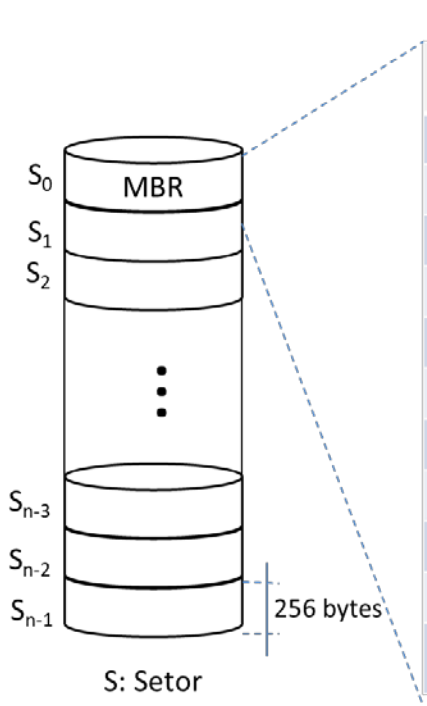
Tabela 2 – Estrutura interna de um *i-node* T2FS (estrutura *t2fs_inode*)

Posição relativa	Tamanho (bytes)	Nome	Descrição
0	4	*blocksFileSize*	Tamanho do arquivo expresso em número de blocos. Se o registro fizer referência a um *softlink* este valor será sempre 0x01 (um).
4	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. Se o registro fizer referência a um *softlink* este valor corresponderá ao tamanho do *string*.
8	8	*dataPtr[2]*	Dois ponteiros diretos.
16	4	*singleIndPtr*	Ponteiro de indireção simples.
20	4	*doubleIndPtr*	Ponteiro de indireção dupla.
24	4	*RefCounter*	Contador usado para indicar quantos arquivos referenciam o mesmo *i-node* (*hardlinks*).
28	4	reservado	Não usados

*TODOS os valores são fornecidos em formato *little-endian*

3 Disco T2FS: formatação lógica

O disco físico do T2FS será simulado a partir de um arquivo de dados denominado de *t2fs_disk.dat* fornecido pelos professores. Esse arquivo simulará, como ocorre nas máquinas virtuais, um disco virtual. Esse disco virtual estará formatado fisicamente em **setores** de 256 bytes, sendo que o primeiro setor do disco – o setor identificado pelo número 0 - é o MBR (*Master Boot Record*) com informações relativas ao disco como partições do disco e tamanho do disco. A Figura 2 ilustra esse disco e o conteúdo do MBR.



Master Boot Record (MBR)

Byte inicial	Tamanho (bytes)	Valor (Hexa)	Descrição
0	2	7E32	Versão do disco/trabalho
2	2	0100	Tamanho do setor em bytes
4	2	8	Byte inicial da tabela de partições
6	2	4	Quantidade de partições no disco
8	4	Setor_i1	Endereço do bloco de setor da partição #0
12	4	Setor_f1	Endereço do último setor da partição #0
16	24	String	Nome da partição #0
40	4	Setor_i2	Endereço do setor de inicio da partição #1
44	4	Setor_f2	Endereço do último setor da partição #1
48	24	String	Nome da partição #1
...	....	....	(repete 3 últimos campos para as duas outras partições #2 e #3)

Figura 2 – Disco *t2fs_disk.dat*

O disco virtual *t2fs_disk.dat* deverá ter uma ou mais partições formatadas logicamente a partir da execução de uma função denominada *format2*, pertencente à API da biblioteca *t2fs*. A formatação lógica prepara a partição para receber arquivos no T2FS. Para isso, a função *format2* insere na partição as estruturas gerenciais necessárias ao sistema de arquivos e define os blocos de disco. A formatação deve criar e inicializar o superbloco (campos fornecidos na tabela 3), calcular e definir a área para os *i-nodes* T2FS, e os *bitmaps* para manter o controle de *i-nodes* e blocos de disco livres e ocupados. As áreas que compõem uma partição T2FS são, nessa ordem:

Superbloco: é a área de controle do sistema de arquivos. Essa área **ocupa o primeiro bloco da partição**, conforme aparece na tabela 3. 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).

Área de *bitmap* de blocos livres/ocupados: formada por 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.

Área de *bitmap* de *i-nodes* livres/ocupados: formada por 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 blocos do disco e *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* T2FS. O tamanho dessa área, em número de blocos, é dado por *inodeAreaSize*, fornecido no superbloco. Essa área deve ocupar 10% dos blocos da partição.

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.

Tabela 3 – 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*	0x7E32	Versão atual do sistema de arquivos: (valor fixo 0x7E3=2019; 2=2º semestre).
6	2	*superblockSize*	1	O superbloco ocupa o primeiro bloco da partição.
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 da partição .
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	4	*Checksum**	*c*	*Checksum* do superbloco
24 até o final		*reservado*		Não usados

TODOS os valores são fornecidos em formato *little-endian*

*Checksum é calculado somando os 20 bytes iniciais do superbloco, agrupados de quatro em quatro (portanto, inteiros sem sinal), e complementando esse valor (complemento de um). Se o superbloco for válido, o valor resultante dessa soma complementada deverá ser igual ao valor presente no campo *checksum*. Na formatação da partição, a função "format2()" deverá preencher, adequadamente, o campo *Checksum*.

4 Interface de Programação da T2FS (libt2fs.a)

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

As funções a serem implementadas estão resumidas na tabela 4. 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 erro, de acordo com o *prototype* da função.

A implementação do sistema de arquivos T2FS deve ser feita de tal forma que seja possível ter até 10 (dez) arquivos regulares abertos simultaneamente. Notar que se pode abrir um 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.
`int format2 (int partition, int sectors_per_block)`	Formata logicamente a partição de número "partition", conforme informada no MBR do disco *t2fs_disk.dat* para o sistema de arquivos T2FS, definido o bloco de dados no disco com um tamanho corresponde ao múltiplo de setores dado por *sectors_per_block*. Deve ser reservado 10% dos blocos da partição para conter os *i-nodes*. Se necessário, arredondar para cima.
`int mount(int partition)`	Monta a partição identificada por *partition* no ponto de montagem "/" (raiz do sistema de arquivos)
`int umount()`	Desmonta a partição atualmente montada, liberando o ponto de montagem "/" (raiz do sistema de arquivos)
`FILE2 create2 (char *filename)`	Função usada para criar um novo arquivo no disco e abri-lo, sendo, nesse último aspecto, equivalente a função *open*2. No entanto, diferentemente da função *open2* (ver abaixo), se *filename* referenciar um arquivo já existente, o mesmo terá seu conteúdo removido e assumirá um tamanho de zero bytes.
`int delete2 (char *name)`	Função usada para remover (apagar) um arquivo do disco. O parâmetro "name" pode se referir a um arquivo ou a um link.
`FILE2 open2 (char *name)`	Função que abre um arquivo existente no disco. O parâmetro "name" pode se referir a um arquivo ou a um link.
`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. As leituras são feitas a partir da posição atual do ponteiro corrente do arquivo. Após a leitura, o ponteiro corrente deve ser atualizado para a nova posição.
`int write2 (FILE2 handle, char *buffer, int size)`	Função usada para realizar a escrita de uma certa quantidade de bytes (*size*) de um arquivo. As escritas são feitas a partir da posição atual do ponteiro corrente do arquivo. Após a escrita, o ponteiro corrente deve ser atualizado para a nova posição.
`int opendir2 (void)`	Função que abre o diretório raiz
`int readdir2 (DIRENT2 *dentry)`	Função usada para ler as entradas do diretório raiz.
`int closedir2 (void)`	Função usada para fechar o arquivo diretório raiz.
`int sln2(char char *linkname, char *filename)`	Função que cria um link simbólico (*softlink*) com o nome dado por *linkname*, para um arquivo fornecido por *filename*.
`int hln2(char char *linkname, char *filename)`	Função que cria um link estrito (*hardlink*) com o nome dado por *linkname*, para um arquivo fornecido por *filename*.

5 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 do disco, que serão endereçados através de sua numeração sequencial a partir de zero. Os setores 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`" 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`" 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 a função format2() deverá instalar 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.

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*"
- Relatório final de entrega (arquivo excel, preenchido e impresso em pdf)

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 com os programas usados para testar a implementação, ou seja, os executáveis dos programas de teste. É aqui que reside o fonte do *t2shell*.
	include	DIRETÓRIO: local onde são postos todos os arquivo ".h". Nesse diretório deve estar o "*t2fs.h*", "t2disk.h", "*apidisk.h*" e o "*bitmap2.h*"
	lib	DIRETÓRIO: local onde será gerada a biblioteca "*libt2fs.a*". (junção da "*t2fs*" gerado por vocês com "*apidisk.o*" e com o "*bitmap2*.o"). Os binários *apidisk.o* e *bitmap2*.o também será armazenado neste diretório.
	src	DIRETÓRIO: local onde são postos todos os arquivos ".c" (códigos fonte) usados na implementação do T2FS. É aqui que será fornecido o arquivo "t2fs.c", a ser usada como ponto de partida para o desenvolvimento do trabalho. Esse arquivo tem todas as funções da interface, implementadas apenas com o retorno de um código de erro.
	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

7 Avaliação

Somente serão avaliados os trabalhos que obedecerem às seguintes condições:

- Entrega da planilha com o relatório final do trabalho;
- Obediência à especificação (formato e nome das funções);
- Compilação e geração da biblioteca **sem erros ou *warnings***;
- Entrega de todos os arquivos solicitados conforme organização de diretórios fornecidos na seção 6;
- Execução correta dentro da máquina virtual *alunovm-sisop.ova*.

A avaliação dos trabalhos será realizada pela aplicação de um conjunto de programas padronizados desenvolvidos pelos professores da disciplina. A nota será proporcional à quantidade de execuções desses programas, usando a API *libt2fs* e seguindo a especificação, considerando-se a dificuldade relativa de cada um.

8 Avisos Gerais – LEIA com MUITA ATENÇÃO!!!

1. É condição para avaliação dos trabalhos a obediência RÍGIDA dos padrões de entrega definidos na seção 6.
2. O trabalho deverá ser desenvolvido em grupos de DOIS ou TRÊS componentes. NÃO serão aceitas outras composições de grupo.
3. O trabalho final deverá ser entregue até a data prevista, conforme cronograma de entrega no Moodle. Deverá ser entregue um arquivo *tar.gz* conforme descrito na seção 6. NÃO haverá extensão de prazos.

9 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. 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
```