Trabalho Prático II – ENTREGA: 19 de Novembro 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 uma variação de alocação encadeada para a criação de arquivos e diretórios.

Esse Sistema de Arquivos será chamado, daqui para diante, de T2FS (*Task 2 – File System – Versão 2018.2*) 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 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).

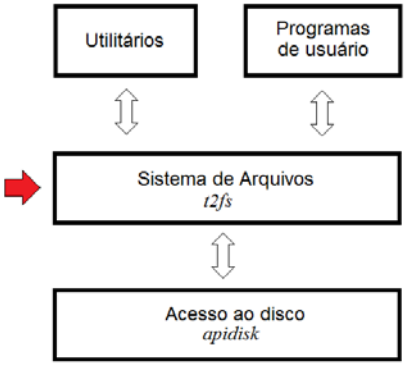


Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

2 Estrutura de um volume T2FS

O espaço disponível no disco formatado logicamente para T2FS (volume ou partição) está dividido em três áreas: superbloco, FAT (*File allocation Table*) e área de dados, como pode ser visto na Figura 2.

- Superbloco: contido no setor lógico zero, fornece as informações relativas à organização de disco do sistema de arquivos T2FS;
- *File Allocation Table* (FAT): localizada a partir do setor lógico 1 e ocupando *k* setores lógicos do disco. Usada para controlar a alocação dos *clusters* e o encadeamento de clusters que formam arquivos.

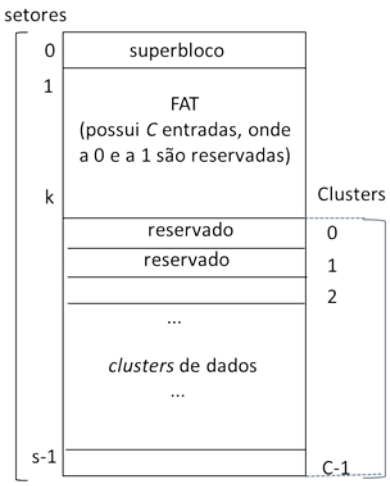
- Área de dados: inicia a partir do primeiro setor livre após a FAT e corresponde a área onde os dados dos arquivos serão armazenados. Essa área estende-se até o final do disco, sendo organizada em *clusters*, que são conjuntos contíguos de *n* setores lógicos.

A quantidade de setores lógicos ocupados pela FAT depende do tamanho da partição (volume) T2FS, do tamanho do setor lógico e da quantidade de *clusters* da área de dados. Essas informações são fornecidas no superbloco e especificam a geometria (formato) do disco T2FS, conforme fornecido na Tabela 1. Os setores lógicos possuem um tamanho fixo de 256 bytes.

Os setores da área de dados do disco estão agrupados em *C clusters* cuja numeração inicia com o valor zero. Assim, a área de dados é composta por *C clusters* identificados de 0 *a C-1*. Os *clusters* de números 0 e 1 são reservados para controle do próprio T2FS. Os números de identificação dos *clusters* possuem 32 bits. O número de setores contíguos que formam um *cluster* é informado no campo *SectorsPerCluster* do superbloco (ver Tabela 1).

Tabela 1 – Descrição dos campos do superbloco (armazenados em formato *little-endian*)

Posição relativa	Tamanho (bytes)	Nome	Valor	Descrição
0	4	*id*	"T2FS"	Identificação do sistema de arquivo. É formado pelas letras "T2FS".
4	2	*version*	0x7E22	Versão atual desse sistema de arquivos Valor fixo 0x7E2=2018; 2=2º semestre.
6	2	*SuperBlockSize*	0x01 0x00	Quantidade de setores lógicos que formam o superbloco. (fixo em 1 setor)
8	4	*DiskSize*		Tamanho total, em bytes, da partição T2FS. Inclui o superbloco, a área de FAT e os *clusters* de dados.
12	4	*NofSectors*		Quantidade total de setores lógicos da partição T2FS. Inclui o superbloco, a área de FAT e os *clusters* de dados. (Corresponde à *s* na figura 2).
16	4	*SectorsPerCluster*		Número de setores lógicos que formam um *cluster*.
20	4	*pFATSectorStart*	0x01 0x00	Número do setor lógico onde a FAT inicia. (fixo no setor 1)
24	4	*RootDirCluster*		Cluster onde inicia o arquivo correspondente ao diretório raiz.
28	4	*DataSectorStart*		Primeiro setor lógico da área de blocos de dados (*cluster 0*). (Corresponde ao setor "k" na figura 2).
32 até o final (*255*)		*reservado*		Não usados.



Um disco T2FS é formado por *s* setores, onde o setor *0* corresponde ao superbloco, os setores de *1* a *k* são destinados ao armazenamento da FAT. O restante do disco é formado por *C clusters*, onde os *clusters* 0 e 1 são reservados. A FAT possui *C* entradas, uma para cada *cluster* do disco.

Figura 2 – Organização do disco lógico T2FS

2.1 Hierarquia de arquivos no T2FS

O T2FS é um sistema de arquivos que emprega um diretório em grafo e utiliza o método de alocação encadeada para organização de arquivos e gerenciamento de espaço do disco. Por possuir uma hierarquia em grafo, os arquivos T2FS podem ser especificados através de seus caminhos de forma absoluta ou relativa (*pathname*). Para criar um grafo são usados links simbólicos (*softlinks* ou atalho).

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:

- o caractere "." indica o diretório corrente;
- a sequência de caracteres ".." indica o diretório-pai.

Portanto, 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.

Os arquivos do T2FS possuem nomes formados por até 54 caracteres alfanuméricos (0-9, a-z e A-Z). Os nomes são *case-sensitive*.

Um *link* simbólico T2FS é implementado através de um arquivo com apenas UM bloco, cujo conteúdo é um string C que fornece o caminho absoluto ou relativo (*pathname*) para o objeto (arquivo ou diretório) para qual o *link* faz referência. Não são permitidos nomes de atalhos (*links* simbólicos) com uma quantidade de caracteres que exceda o tamanho do bloco, menos uma unidade. Lembre-se que os strings em C possuem o caracter '\0' como terminador.

2.2 Organização e gerenciamento de espaço em disco do T2FS

O tipo de encadeamento do T2FS é aquele em que os ponteiros de encadeamento estão em uma área à parte, denominada de *File Allocation Table* (FAT), similar a do sistema operacional MS-DOS.

Há dois tipos de arquivos no T2FS: regulares e diretórios. Para todos os arquivos existentes no T2FS existe um registro chamado de *entrada de diretório*. Essas entradas têm exatamente o mesmo formato no diretório raiz ou em subdiretórios. Um arquivo regular é aquele que contém dados de um usuário, podendo ser um arquivo texto (caracteres ASCII) ou binário. Um arquivo de diretório é aquele que contém informações sobre outros arquivos regulares, diretórios ou *links*, sendo formado por entradas de diretório. O T2FS implementa o conceito de link

simbólico, ou seja, é possível criar um nome alternativo (alias ou atalho) para um arquivo regular ou para um diretório. Dessa forma, em um arquivo do tipo diretório, incluindo o diretório raiz, há três tipos de entradas: arquivos regulares, diretórios e *links* simbólico.

Sempre que um arquivo regular for criado, deve ser ocupada uma *entrada de diretório* no diretório informado pelo caminho (relativo ou absoluto), e alocado um primeiro *cluster* para o arquivo. À medida que o arquivo regular recebe dados, devem ser alocados e encadeados os *clusters* necessários ao armazenamento desses dados. Sempre que um arquivo tiver sua quantidade de dados reduzida e, em consequência dessa redução, *clusters* forem liberados, o encadeamento na FAT deve ser atualizado. Os *clusters* liberados devem ser tornados livres. Na remoção de um arquivo do disco T2FS, todos os *clusters* que lhe pertenciam devem ser liberados e a entrada do diretório deve ser marcada como livre (ver Tabela 3, seção 2.4).

Toda vez que um diretório (subdiretório) for criado, deve ser ocupada uma *entrada de diretório* no diretório em que se está criando esse subdiretório, e alocado um *cluster*. Além disso, deve-se acrescentar no diretório recém criado entradas para os diretórios "." (ponto) e ".." (ponto-ponto). A remoção de um diretório só é possível se o diretório estiver vazio, isso é, não possuir nenhuma entrada válida à exceção dos registros "." (ponto) e ".." (ponto-ponto). A remoção de um diretório implica na liberação do cluster que ocupava e a entrada do diretório deve ser marcada como livre.

Por sua vez, na criação de um *link* simbólico, também deve ser ocupada uma *entrada de diretório* no diretório em que se está criando esse *link*, e alocado um cluster. O conteúdo a ser posto no cluster de um *link* simbólico é o caminho (relativo ou absoluto) para o local do arquivo ao qual esse *link* faz referência. Por ser do tipo simbólico, sempre que um *link* for removido deve-se liberar a entrada de diretório que descreve o *link* e o *cluster* associado ao *link*. Lembrando que, ao apagar um *link* simbólico, apenas o link é removido, não o arquivo que ele aponta. Por outro lado, se o arquivo regular ou diretório apontado pelo *link* for removido, o *link* ficará inconsistente, pois apontará para um recurso inexistente. Esse "dangling" de link só será identificado caso o link seja utilizado.O tamanho máximo de um *link*, em quantidade de caracteres, não deve ultrapassar o tamanho de um cluster menos uma unidade (reservado para o caractere \0, terminador de *strings* em C).

2.3 *File Allocation Table* (FAT)

Os setores lógicos, iniciando com o setor lógico definido por *pFATSectorStart* (Tabela 1) são destinados a armazenar a tabela de alocação e encadeamento dos *clusters* de dados. O tamanho exato, em setores lógicos, ocupado pela FAT depende da quantidade de setores lógicos que compõe um *cluster* e do tamanho total da partição T2FS.

A FAT é organizada como uma lista com *C* elementos de 32 bits (4 bytes). Cada elemento dessa lista está associado a um *cluster* de dados. Portanto, a quantidade de elementos válidos nessa lista é igual à quantidade total de *clusters* da área de dados. Cada elemento da FAT pode receber um valor conforme a tabela 2. As entradas 0 e 1 da FAT são reservadas e, por consequência, os clusters 0 e 1 existem no disco, mas não podem ser usados (reservados pelo T2FS).

Tabela 2 – Números válidos para entradas na FAT e sua interpretação (formato little-endian)

Valor (4 bytes)	Significado
0x00000000	O *cluster* associado está livre.
0x00000001	Valor que não pode aparecer nos elementos da FAT.
0x00000002 até 0xFFFFFFFD	Número do próximo *cluster* do arquivo. Esses valores são usados para indicar o encadeamento dos *clusters* do arquivo.
0xFFFFFFFE	*Cluster* com algum setor lógico com defeito (*bad sector*). Não pode ser alocados para armazenamento de dados.
0xFFFFFFFF	Último *cluster* do arquivo (EOF – *end of file*).

Notar que:

1. Um *cluster* livre é identificado pelo valor 0x00000000 na sua entrada corresponde na FAT;
2. A entrada da FAT correspondente ao último *cluster* de um arquivo recebe o valor 0xFFFFFFFF (*end-of-file*).
3. Um *cluster* que possui algum setor lógico com defeito estará identificado pelo valor 0xFFFFFFFE na sua entrada correspondente da FAT. Ao procurar um *cluster* livre para criar, ou expandir, um arquivo, o sistema de arquivos não poderá alocar um *cluster* que tenha um setor defeituoso.
4. Os clusters de dados que formam um arquivo podem ser indicados por valores que vão desde 0x00000002 até 0xFFFFFFFD. Esse valores são colocados nos elementos correspondentes na FAT.

2.4 Implementação de Diretórios no T2FS

Conforme especificado anteriormente, a hierarquia de arquivos do T2FS segue uma organização em grafo, permitindo definir subdiretórios dentro de diretórios e assim sucessivamente. Portanto, um diretório T2FS pode conter registros de:

- Arquivos regulares;
- Arquivos de diretórios (subdiretórios)
- *Links* simbólicos (*softlinks*) para arquivos e diretórios;

Cada arquivo existente em um disco formatado em T2FS possui uma entrada (registro) em um diretório. Todos os diretórios, além de entradas para diretórios, arquivos e *links*, devem possuir duas entradas especiais de diretório: a entrada "." (ponto) e a entrada ".." (ponto ponto), para indicar o próprio diretório (diretório corrente) e o seu diretório pai, respectivamente.(0 diretório raiz é um caso especial, detalhado no item 2.5).

Os diretórios T2FS são arquivos com duas características especiais. Primeira, todo diretório T2FS ocupa exatamente UM *cluster*, ou seja, ele tem tamanho fixo. Portanto, sempre que um diretório (subdiretório) T2FS for criado, deverá ser alocado um *cluster*. Isso implica que um diretório terá uma quantidade máxima de entradas que dependerá da quantidade de setores que formam o *cluster*. Um erro deverá ser gerado caso se tente criar mais arquivos que o número de entradas disponíveis em um diretório.

Segunda, internamente, os diretórios T2FS organizam as entradas como uma lista linear de registros de tamanho fixo. Cada registro é uma entrada do diretório e está associada a um arquivo T2FS. A tabela 3 mostra a estrutura de um registro T2FS (estrutura *t2fs_record*), onde todos os valores numéricos estão armazenados em formato *little-endian*.

Tabela 3 – 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. • 0x03, *link* simbólico
1	51	*name*	Nome do arquivo: *string* com caracteres ASCII (0x21 até 0x7A), *case sensitive*. Caracteres não utilizados devem ser preenchidos com o caractere especial "\0" (0x00).
52	4	*bytesFileSize*	Tamanho do arquivo. Expresso em número de bytes. Notar que o tamanho de um arquivo não é um múltiplo do tamanho dos *clusters* de dados. Portanto, o último *cluster* 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 arquivo do tipo diretório ou a um *link* simbólico este valor deve corresponder ao tamanho do bloco.

Posição relativa	Tamanho (bytes)	Nome	Descrição
56	4	*clustersFileSize*	Tamanho do arquivo, expresso em número de *clusters*. Se o registro fizer referência a um arquivo do tipo diretório ou a um *link* simbólico este valor será sempre 0x00000001 (um).
60	4	*firstCluster*	Número do primeiro *cluster* de dados correspondente a essa entrada de diretório

2.5 Diretório raiz

O diretório raiz segue exatamente a mesma estrutura de um arquivo de diretório T2FS, conforme descrito na seção 2.4, ou seja, é composto por UM único *cluster* que forma o arquivo diretório e possui em seu interior uma lista linear de registros de tamanho fixo (Tabela 3).

No entanto, por ser o primeiro diretório da hierarquia, o diretório raiz apresenta duas exceções. Primeira, a localização do cluster inicial do arquivo raiz é dado pela entrada *RootDirCluster* do superbloco (ver Tabela 1). 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.

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 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-se até 10 (dez) arquivos regulares abertos simultaneamente. Notar que pode-se 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 e abrí-lo, sendo, nesse último aspecto, equivalente a função *open*2.No entanto, diferentemente da *open2*, se *filename* referenciar um arquivo já existente, o mesmo terá seu conteúdo removido e assumirá um tamanho de zero bytes.
`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.

Nome	Descrição
`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.
`int ln2(char *linkname, char *filename)`	Função usada para criar um caminho alternativo (softlink) com o nome dado por *linkname* (relativo ou absoluto) para um arquivo ou diretório fornecido por *filename*.

Obs.: as funções *open2, delete2, opendir2*, *rmdir2 e chdir2* podem receber como parâmetro um *link* simbólico.

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 (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 DOIS ou TRÊS componentes. O trabalho não poderá ser feito individualmente.
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. 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 de disco

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 de disco do T2FS. Dessa forma, um bloco de disco T2FS é formado por uma sequência contígua de *n* setores lógicos. A figura C.1 ilustra esses conceitos de forma genérica.

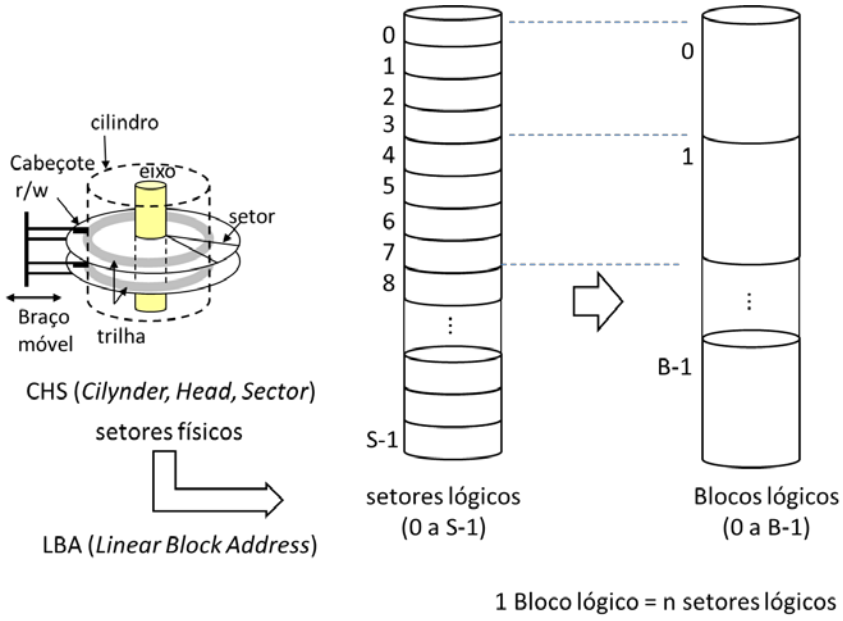


Figura C.1 – setores físicos, setores lógicos e blocos de disco (diagrama genérico)