Trabalho Prático II – ENTREGA: 20 de junho de 2019

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 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. 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 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).

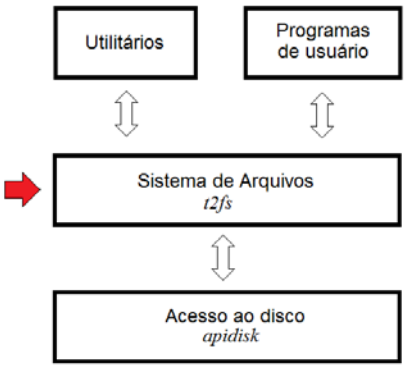


Figura 1 – Componentes principais do T2FS: aplicativos, sistema de arquivos e acesso ao disco.

2 Projeto do T2FS

Cada grupo, a partir das opções fornecidas na Tabela 1, deverá escolher as funcionalidades para a sua versão do T2FS. Dessa forma, a primeira tarefa a ser desenvolvida pelo grupo será definir as características da sua versão de T2FS. No entanto, há um conjunto mínimo de requisitos que deverá ser atendido, como a existência de arquivos regulares (binários ou ASCII), diretórios e vínculos.

Cada arquivo existente em um disco formatado em T2FS possui uma entrada (registro) em um diretório. Os metadados (atributos) que deverão existir na entrada dependerão das opções para o T2FS escolhidas pelo grupo.

Como o T2FS será um sistema de arquivos que, dependendo da sua escolha, empregará uma hierarquia de diretórios em árvore ou em grafo, os arquivos T2FS poderão 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 será a mesma do Unix, isso é:

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

Portanto, se o sistema de arquivos suportar caminhos relativos, um arquivo no diretório corrente poderá 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.

O T2FS deverá suportar arquivos com nomes formados por até 31 caracteres alfanuméricos (0-9, a-z e A-Z). Os nomes são *case-sensitive*.

Se a sua opção contemplar *links* simbólicos, o mesmo deverá ser implementado no T2FS através de um arquivo com apenas UM bloco. Não serão permitidos nomes de atalhos (*links* simbólicos) com uma quantidade de caracteres que exceda o tamanho de um bloco de dados, menos uma unidade. Lembre-se que os *strings* em C possuem o caracter '\0' como terminador.

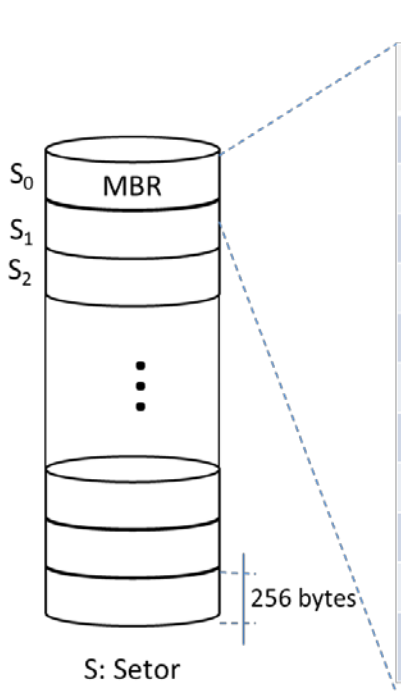
Tabela 1 – Opções para definição de características do T2FS

Característica de projeto	Opções (escolher UMA para cada característica)
Alocação do espaço em disco	- Contígua pura - Encadeada pura - Encadeada estilo FAT - Indexada pura - Indexada com encadeamento de blocos de índice - Indexada com blocos de índice multinível - Indexada combinada (estilo *i-nodes*)
Gerência de espaço livre	- Bitmaps (ou vetor de bits) - Lista encadeada - Agrupamento - Contagem
Hierarquia de diretórios T2FS	- Árvore - Grafo acíclicos
Localização do diretório raiz	- Região fixa do disco - Indicação explícita do local do arquivo diretório raiz
Registro de diretório	- tamanho fixo - tamanho variável
Organização interna dos registros de diretório	- organização linear - tabela de dispersão (hash) - em árvore
Arquivos	- arquivos regulares (binário e ASCII)
Vínculos (*links*)	- vínculos simbólicos (*softlinks*) - vínculos estritos (*hardlinks*) - vínculos simbólicos e vínculos estritos
Caminho	- relativo - absoluto - relativo e absoluto

ATENÇÃO: não será permitido que dois, ou mais grupos, tenham exatamente o mesmo conjunto de escolhas.

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, por exemplo, 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	7E31	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	LBA_i1	Endereço do bloco de inicio da partição #1
12	4	LBA_f1	Endereço do último bloco da partição #1
16	24	String	Nome da partição #1
40	4	LBA_i2	Endereço do bloco de inicio da partição #2
44	4	LBA_f2	Endereço do último bloco da partição #2
48	24	String	Nome da partição #2
...	....	....	(repete 3 últimos campos para as duas outras partições #3 e #4)

LBA = endereços (em hexa) dos blocos lógicos

Figura 2 – Disco *t2fs_disk.dat*

O disco virtual *t2fs_disk.dat* deverá ser formatado logicamente a partir da execução de uma função denominada *format2*, pertencente a API da biblioteca t2fs.

A execução da função *format2* formatará logicamente o disco virtual com as estruturas gerenciais necessárias (e coerentes) com a escolha das opções do grupo para o T2FS como, por exemplo, aquelas usadas para gerenciar os blocos livres do disco.

4 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 1. A implementação de seu trabalho deve possuir TODAS AS FUNÇÕES aqui especificadas, **mesmo que não tenham sido implementadas ou que suas escolhas de opções não a contemple**. 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 2 – Interface de programação de aplicações – API - da *libt2fs*

Nome	Descrição
`int format2 (int sectors_per_block)`	Formata logicamente a partição 0 (zero) informada no MBR do disco *t2fs_disk.dat* para o sistema de arquivos T2FS, definido usando blocos de dados de tamanho corresponde a um múltiplo de setores dado por *sectors_per_block*.
`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.
`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.

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 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.

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 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*" e o "*apidisk.h*"
	lib	DIRETÓRIO: local onde será gerada a biblioteca "*libt2fs.a*". (junção da "*t2fs*" gerado por vocês com "*apidisk.o*"). O binário *apidisk.o* também será armazenado 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

7 Avaliação

A avaliação do trabalho considerará as seguintes condições:

- Nota atribuída a apresentação presencial de um pitch, de duração de 6 min, por TODOS os membros do grupo.
- 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;
- Entrega de um relatório final conforme formulário disponibilizado no moodle.
- Execução correta dentro da máquina virtual *alunovm-sisop.ova.*

Itens que serão avaliados e sua valoração:

- 20,0 pontos: relativo à apresentação do *pitch*. Essa nota terá uma componente do grupo (10 pontos) e uma componente individual (10 pontos) de acordo com a participação de cada membro no *pitch*.
- 10,0 pontos: uma nota de auto-avaliação. Essa nota será dada pelo próprio membro do grupo levando em conta seu aprendizado, sua participação e envolvimento no trabalho do grupo.
- 10,0 pontos: uma nota de avaliação entre os pares, ou seja, cada um dos membros do grupo deverá dar uma nota para cada um dos outros membros. Essa nota deverá levar em conta a participação e o envolvimento do colega na realização do trabalho e será feita de forma anônima.
- 60,0 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.

8 O que é um pitch? O que é necessário fazer?

Um *pitch*, também chamado de *discurso de elevador*, de acordo com a wikipedia, “remete a ideia de uma fala ou diálogo breve e objetivo que um indivíduo utiliza para discursar a respeito de um produto, serviço ou uma organização, demonstrando seus benefícios e valores, despertando o interesse do interlocutor. A analogia ao elevador se deve ao fato de que o tempo gasto para esse discurso deve durar de trinta segundos a dois minutos, o que dura uma curta "viagem" ou deslocamento utilizando um elevador comum.” Os *pitches* se tornaram comuns com o crescimento da área de empreendedorismo, onde os empreendedores têm poucos minutos para apresentar e convencer investidores que a sua ideia, ou *startup*, merece um aporte financeiro. Os *pitches* costumam ter durações de 1 minuto, 3 minutos ou 5 minutos. Em rodadas mais avançadas de negociação com investidores, o tempo pode aumentar, mas não fica muito superior a uma dezena de minutos. Ao esgotar o tempo do *pitch*, a apresentação é interrompida com aplausos dos investidores. Por isso, é importantíssimo que o discurso caiba exatamente no tempo previsto, não ultrapassando, nem ficando muito abaixo do tempo disponível.

Na definição do T2FS, cada grupo terá SEIS MINUTOS, cravados, para apresentar as suas decisões de projeto, isso é, as opções da Tabela 1. A ideia é que cada grupo “venda” a sua versão do T2FS, apresentando para os colegas as funcionalidades oferecidas pelo seu sistema de arquivos. Alguns pontos importantes:

- O *pitch* será apresentado por todos os grupos no dia 23 de maio e será composto por uma apresentação power point que deverá ser entregue, via moodle, até a meia noite de dia 22 de MAIO.
- O *pitch* deverá respeitar os seis minutos máximos. Treinem a apresentação para caber nesse tempo.
- Todos os membros do grupo devem se envolver com o *pitch*. Para um grupo de 4 pessoas, o tempo médio de fala é de 1 minuto e meio. (Sim, a apresentação ficará na forma de um jogral).
- Itens obrigatórios a serem colocados na apresentação do pitch:
    - Nome fantasia para identificar o grupo, como se fosse uma marca;
    - Nomes dos membros do grupo e identificação clara do gerente;
    - Um slide com uma tabela similar a tabela 1 informado a opção do grupo;
    - Um slide com uma figura do disco T2FS após a sua formatação lógica, indicando os componentes ou áreas desse disco, de acordo com as opções do grupo;
- O que for apresentado no *pitch* será considerado como o produto a ser entregue pelo grupo. NENHUMA alteração posterior nas opções de projeto será aceita. Funcionalidade prometida e não realizada implicará em desconto na nota final.

E, não esqueçam, um *pitch* deve ser BREVE e OBJETIVO e também deve fornecer os benefícios e funcionalidades do produto que está sendo “vendido” ao seu interlocutor.

9 Avisos Gerais – LEIA com MUITA ATENÇÃO!!!

1. O trabalho deverá ser desenvolvido em grupos de até QUATRO componentes. O trabalho não poderá ser feito individualmente.
2. 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.

10 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.

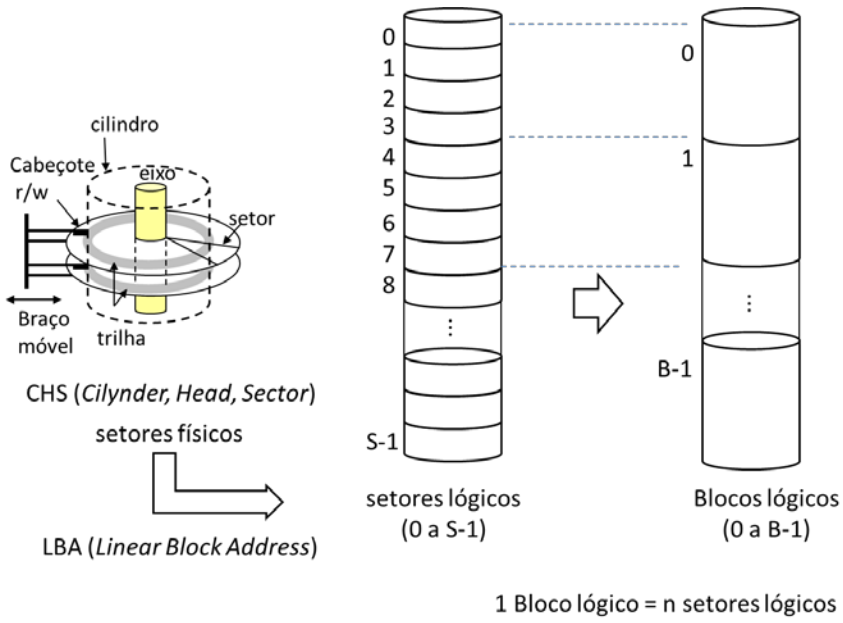


Figura C.1 – setores físicos, setores lógicos e blocos de disco (diagrama genérico)