Sistemas Operacionais – Profa. Roberta Lima Gomes

Laboratório 6 - File System Unix

Estrutura de Diretórios

Os principais sistemas de arquivos usados para a formatação de discos locais em Linux são o ext2, ext3, ext4, reiser, xfs e jfs, entre outros. Mas em geral, os diretórios de um sistema de arquivos no UNIX têm uma estrutura pré-definida comum, com poucas variações:

bin - contains system binary executables
boot - contains files necessary for the system to boot up
dev - contains device files which function as an interface to the various hardware drivers. These will vary greatly depending on the version of Unix.
etc - contains system configuration settings
home - contains the user's home directories (often but not at LSC/ATM Linux network)
mnt/homes - contains the user's home directories at the LSC/ATM network)
lib - contains system libraries for 32- bit applications
lib64 - contains system libraries for 64- bit applications
mnt - used to hold mount point directories for removable storage
opt - contains optional applications
proc - contains a virtual file system which holds information about running processes and the state of the system.
root - the root (administrator) user's home directory
sbin - contains static binary executables needed for the system
tmp - temporary directory used by many applications
usr - contains binaries, data and settings for various applications. The structure of /usr mimics the root file system organization.
var - stores logs, data for services and other transient data.

TAREFAS:

1) No terminal, vá até o diretório **`/proc`** e liste seu conteúdo (**`ls -l`**). Observe que os subiretórios corespondem aos PIDs dos processos correntes (execute **`ps -lax`** e verifique isso).

O /proc é, por vezes, chamado de "pseudo sistema de arquivos de informações de processos" ou process information pseudo-file system. O diretório não contém "arquivos de verdade", mas as informações referentes ao seu sistema em tempo de execução (runtime).

Entre as informações disponíveis no /proc, você pode encontrar a quantidade de memória presente no sistema, os dispositivos de armazenamento que estão montados, a configuração atual do hardware, o tempo que o seu dispositivo está ligado etc.

2) Agora entre no subdiretório cujo nome seja o pid da sua bash (execute **`ps`** para ver o PID da bash). Ali você encontrará várias informações sobre este processo… consulte algumas dessas informações para a sua bash :

```
more /proc/PID/cmdline  // Argumentos da linha de comando.
more /proc/PID/maps     // Mapas de memória para os executáveis e arquivos da
                        // biblioteca.
```

`more /proc/PID/stat` // Infos gerais de estado do processo
// Segue descrição de alguns valores printados:

```
(1) pid  %d
          The process ID.

(2) comm  %s
          The filename of the executable, in parentheses.
          This is visible whether or not the executable is
          swapped out.

(3) state  %c
          One of the following characters, indicating process
          state:

          R  Running

          S  Sleeping in an interruptible wait

          D  Waiting in uninterruptible disk sleep

          Z  Zombie

          T  Stopped (on a signal) or (before Linux 2.6.33)
             trace stopped

(14) utime  %lu
          Amount of time that this process has been scheduled
          in user mode, measured in clock ticks (divide by
          sysconf(_SC_CLK_TCK)).  This includes guest time,
          guest_time (time spent running a virtual CPU, see
          below), so that applications that are not aware of
          the guest time field do not lose that time from
          their calculations.

(15) stime  %lu
          Amount of time that this process has been scheduled
          in kernel mode, measured in clock ticks (divide by
          sysconf(_SC_CLK_TCK)).
```

3) Você consegue encontrar o executável do seu SO? Execute `ls -l /` (diretório raiz).

Observe aparece algo assim:

```
lrwxrwxrwx   1 root root        30 jun 29  2019 vmlinuz -> boot/vmlinuz-4.18.0-25-generic
```

Esse 1º. caracter na linha indica o tipo de arquivo. Neste caso temos `l` indicando que é um link.

Tipos de Arquivos

Os tipos de arquivos suportados pelo sistema são:

Arquivos normais: sequências de bytes (texto, binário, executável, etc.)

Diretórios: lista de outros arquivos (nome do arquivo e inode)

Arquivos especiais (dispositivos): interface entre o sistema e dispositivos de entrada e saída; podem ser dispositivos orientados a **c**aractere ou a **b**loco

Links: podem ser Simbólicos (*soft link*: ponteiro para outro arquivo) ou Concretos (*hard link*: atribue mais um nome ao mesmo arquivo que esteja na mesma partição)

Sockets e Pipes: usados para comunicação entre processos (mecanismo para programação)

Arquivos de Dispositivos

No UNIX, tudo é apresentado na forma de arquivos. Ao plugar um pendrive no computador, por exemplo, um arquivo será criado dentro do diretório /dev e ele servirá como interface para acessar ou gerenciar o drive USB. Nesse diretório, você encontra caminhos semelhantes para acessar terminais e qualquer dispositivo conectado ao computador, como o mouse e até modems.

TAREFA:

4) No terminal, vá até o diretório **`/dev`** e liste seu conteúdo (**`ls -l`**). Observe que o inicio de cada linha printada indica o tipo de arquivo (**`c`**, **`b`** ou **`d`**… eventualmente algum **`l`**).

Exemplos:

disco IDE	*/dev/hda, /dev/hdb, /dev/hdc, /dev/hdd, ...*
disco SCSI/SATA	*/dev/sda, /dev/sdb, /dev/sdc, /dev/sdd, ...*
partições disco IDE 1	*/dev/hda1, /dev/hda2, /dev/hda3, ...*
partições disco SCSI1	*/dev/sda1, /dev/sda2, /dev/sda3, ...*
terminal de controle	*/dev/tty*
terminal serial	*/dev/tty1, /dev/tty2, /dev/tty3, ….*
subdiretório em que são montados os dispositivos USB	*/usb*

Uma curiosidade: existem quatro arquivos na pasta /dev (**`full`**, **`zero`**, **`random`** e o **`null`**), que não correspondem a devices de fato. Você saberia dizer a função de cada um deles?

TAREFA:

5) No terminal, digite:

```
$ echo "Hello World"
```

e depois

```
$ echo "Hello World" > /dev/null
```

*Obs: o caracter especial '**>**' direciona a saída padrão de um processo para o arquivo passado na linha de comando.*

… O que aconteceu com a saída do comando? Nada... certo!? Mas então o que é o **`/dev/null`**? **[RESPONDA NO FORMULÁRIO ONLINE]**

6) No terminal, digite o comando abaixo e observe o resultado.

```
$ echo "Hello world" > /dev/full
```

Você consegue entender /dev/full ?

Descritores de Arquivos e o Redirecionamento de Entrada e Saída padrão

Na útlima tarefa, você usou o caracter especial '>' para redirecionar a saída padrão de um processo para um arquivo. Se você fez o trabalho da nossa disciplina você já entende o que está acontecendo por baixo dos panos. Mas vamos rever aqui alguns conceitos:

- Cada processo possuiu uma Tabela de Descritores de Arquivo, ou File Descripter Table como mostrado na Figura 1. Essa tabela é criada pelo kernel, mas ela "reside" na memória do processo. Para cada "arquivo" aberto que o processo tenha acesso, deve haver (pelo menos) uma entrada dessa tabela apontando para um "Open File Object". Então um **descritor de arquivo** "x" retornado por uma operação *open* nada mais é do que um número inteiro que representa um **índice** na tabela de descritores.
    - Como já vimos, alguns descritores de arquivos têm um significado especial: 0 é STDIN; 1 é STDOUT; 2 é STDERR.
- Na posição indicada pelo descritor de arquivo "x" teremos um ponteiro para uma posição de memória **no kernel** onde está o *Open File Object* criado quando este arquivo foi aberto.
    - *Open File Object* são criados dentro do kernel e alocados em uma tabela, denominada System File Table.
- Cada *Open File Object* representa no fundo uma "seção independente de acesso a um arquivo". O *Open File Object* contém o contexto desta sessão, como o **modo** em que o arquivo foi aberto, o ***offset*** em que a próxima leitura ou escrita deve ocorrer, o **número de descritores** que apontam para ele, e uma **referência para o inode** do arquivo (que foi carregado em memória). A Figura 1 apresenta uma ilustração dessas estruturas de dados nas memórias dos processos e do kernel.
    - Se um mesmo processo faz dois *opens* no mesmo arquivo, são criados dois descritores e dois *Open File Objects* no kernel, com modos de acessos e offsets totalmente independentes. Essa situação é apresentada na Figura 2 (observem que os dois *Open File Objects* referenciam o mesmo inode).

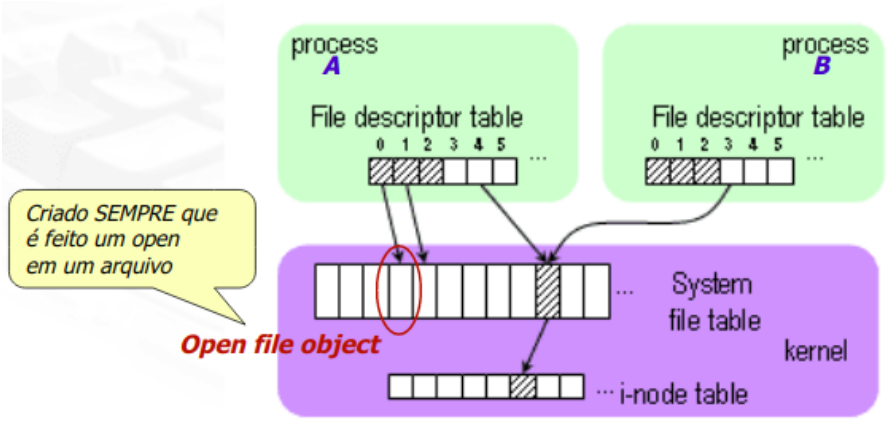

Figura 1: Representação das Tabelas de Descritores e da *System File Table*

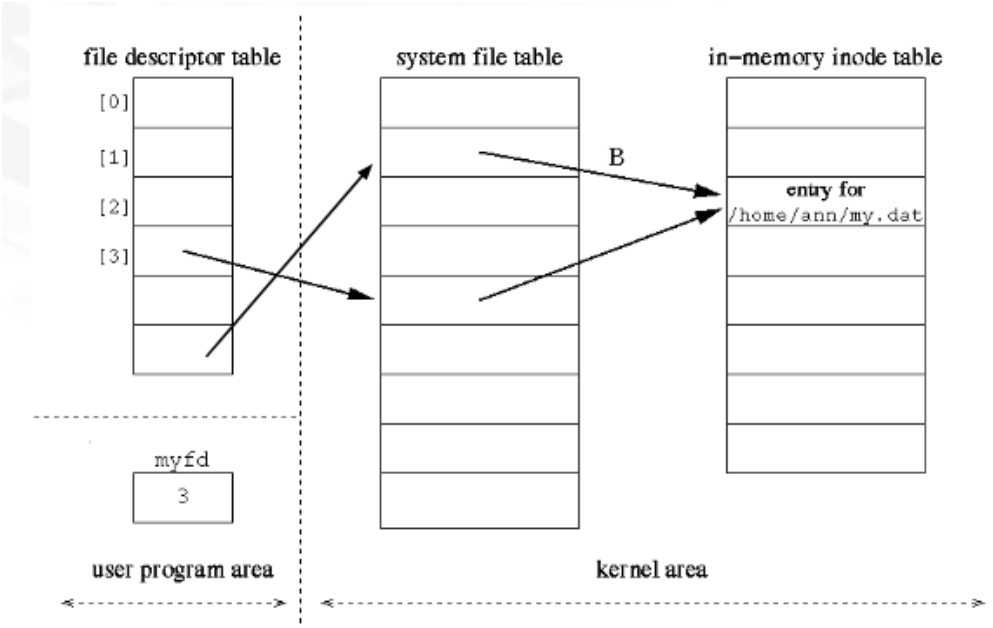

Figura 2: Resultado de dois *opens* sobre o mesmo arquivo.

TAREFA:

7) Considere a Figura 1 novamente. Como seria possível termos o descritor "4" no *Process A* apontando para o mesmo *Open File Object* que o descritor "3" no *Process B*. **[RESPONDA NO FORMULÁRIO ONLINE]**

Voltando então ao comando "**`echo "Hello World" > /dev/null`**". O que acontece é que a bash faz um fork() e em seguida o processo filho :

- antes de fazer o exec, faz um open no arquivo `/dev/null` o que retorna um descritor x;
- em seguida usa o comando `dup2(x,STDOUT)` para copiar o valor contido na posição x da sua Tabela de Descritores para a posição "1". Com isso STDOUT passa a referenciar o *Open File Object* criado para o acesso ao arquivo `/dev/null`.

Inodes e Atributos de Arquivos

Cada arquivo ou diretório possuiu um inode associado.

TAREFA:

8) No terminal, vá até o diretório HOME (**`cd ~`**) e digite **`$ ls -lai .`**

Obs: Na coluna mais à esquerda, você encontra os números do inode de cada arquivo.

Agora faça a mesma coisa de dentro do diretório raiz. Alguém com o inode 1?

Nessa distribuição, provavelmente você deve ver o próprio "/" com inode 2 (lembrando que a primeira linha printada pelo `ls` corresponte ao ".", isto é, o próprio diretório). Mas `/proc` e `/sys` com inode 1. Isso ocorre na verdade porque esses não são diretórios de fato no sistema de arquivos local. Eles são "montados" (veremos isso no finalzinho deste Lab). Mas se você está curioso, digite no terminal o seguinte comando: **`$ findmnt`**

*Vale lembrar que em cada partição de disco, temos um sistema de arquivos independente. Com isso, pode acontecer de arquivos **distintos**, em partições distintas, terem inodes com o **mesmo número**.... afinal, pode existir um inode 10 numa partição A, e um inode 10 em outra partição.*

Como vimos em sala, no inode de cada arquivo estão armazenados diferentes atributos (informações de controle sobre o arquivo). Ali encontramos informações como:

- *Tipo de arquivo:*
  - Ex: regular, diretório, PIPE, links simbólicos, arquivos especiais representando dispositivos
- *Número de hard links* apontando p/ o arquivo (você vai entender o que é ainda neste Lab!)
- *Tamanho* (bytes)
- *Device ID*
- *Número do inode:*
  - Dentro de um mesmo device, um inode tem um número único
- *UIDs e GIDs* do proprietário
- *Timestamps* (último acesso, última modificação e última modificação de atributos)
- *Permissões e mode flags*
  - read, write, execute … Acessos divididos por categorias: owner, group, others

Vale ressaltar que arquivos executáveis têm um atributo especial… o ***suid***. Quando um usuário executa um arquivo, se o *suid* desse arquivo executável estiver setado, o processo criado que irá de fato executar o programa terá seu *effective* UID alterado para o o UID do owner deste arquivo. Um exemplo de executável clássico que tem o *suid* setado o `passwd`.

TAREFA:

9) No terminal, digite o seguinte comando:

```
$  stat /bin/chmod
```

Este é um executável, mas você pode fazer isso para diferentes tipos de arquivos. Observe os campos "Blocos" e "bloco de E/S" (Obs.: podem aparecer em inglês).

10) Agora, digite o seguintes comando e compare sua saída com a do comando da Tarefa 9:

```
$  stat /usr/bin/passwd
```

Observe que ambos chmod e passwd são executáveis cujo owner é root. Mas os usuários comums podem executá-los… isso está dito lá no 1o. campo "Acesso" (ou "Access") listado. Observe as diferenças nesse 1o. campo "Acesso" de cada saída. Marque as opções corretas. **[RESPONDA NO FORMULÁRIO ONLINE]**

Arquivos do tipo Link

O *link* é um mecanismo que faz referência a outro arquivo ou diretório em outra localização. Os links são arquivos especiais e podem ser identificados com um "l" quando executado o comando: "ls -la".

Symbolic links (ou *Soft link)*

No link tipo simbólico, o link é um arquivo especial de disco do tipo link, que tem como conteúdo o caminho para chegar até o arquivo alvo. As principais características são:

- Pode-se fazer links simbólicos em arquivos e diretórios;
- O link simbólico e o arquivo alvo não precisam estar na mesma partição de disco;
- Se o link simbólico for apagado/movido. Somente o link será apagado/movido;
- Qualquer usuário pode criar/desfazer um link simbólico (respeitando as permissões).

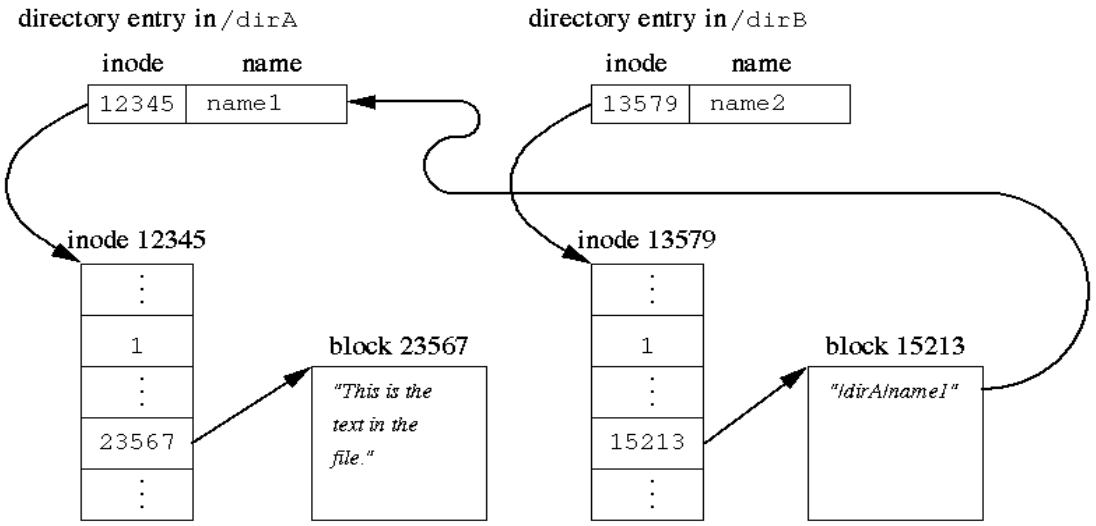

Figura 3: Diagrama ilustrando o arquivo original (à esquerda) um soft link (à direita)

Para criar um link simbólico, podemos usar o seguinte comando no terminal:

```
$ ln –s path1 path2
```

Por exemplo, na Figura 3, o arquivo name2 é um link simbólico criado assim:

```
$ ln –s /dirA/name1 /dirB/name2
```

Também existe uma chamada de sistema que os programadores podem utilizar:

```
int symlink (const char *path1, const char *path2)
    // Cria um link simbólico (path2 -> path1)
```

Mas como o link simbólico é um arquivo especial, se tentarmos dar um cat nele, não veremos o conteúdo texto que existe ali … mas sim veremos o conteúdo do arquivo destino em si. Voltando à Figura 3, se fizermos:

```
$ cat /dirB/name2
```

… veremos na saída

```
This is the text in the file
```

Para ler o conteúdo de fato de um arquivo de tipo link simbólico, devemos usar **`ls -l`** OU **`readlink`**

TAREFA:

11) No terminal, crie um arquivo `teste1` (com algum conteúdo texto dentro) e em seguida crie um link simbólico chamado `teste2` para esse arquivo usando **ln -s** . Depois verifique o resultado usando:

```
$ cat teste2
```

```
$ readlink teste2
```

Por fim, verifique se os inodes dos arquivos `teste1` e `teste2` são iguais ou diferentes (use ls -lai). **[RESPONDA NO FORMULÁRIO ONLINE]**

12) Agora apague o arquivo `teste1` e execute os dois comandos novamente:

```
$ cat teste2
$ readlink teste2
```

O que temos aqui também é chamado de "dangling link"... ou seja, um link que não aponta para nada.

Hard links

No link tipo *hard link*, não temos um arquivo especial. O que acontece é que é criada uma entrada no diretório (em que é criado o *hard link*) que irá referenciar o mesmo inode do arquivo alvo. Sendo assim, teremos duas entradas (nomes de arquivos), no mesmo diretório ou em diretórios diferentes, que correspondem a um mesmo arquivo no disco. As principais características são :

- Não é possível fazer um *hard link* para um diretório;
- Somente é possível fazer *hard link* em arquivos que estejam em uma mesma partição de disco;
- Se o *hard link* for apagado/movido, você estará apagando/movendo a entrada do diretório. O arquivo alvo só é apagado do disco se não houver mais nenhuma outra entrada (*hard link*) referenciando esse inode;
- O usuário deve ter permissão de RW no arquivo destino.

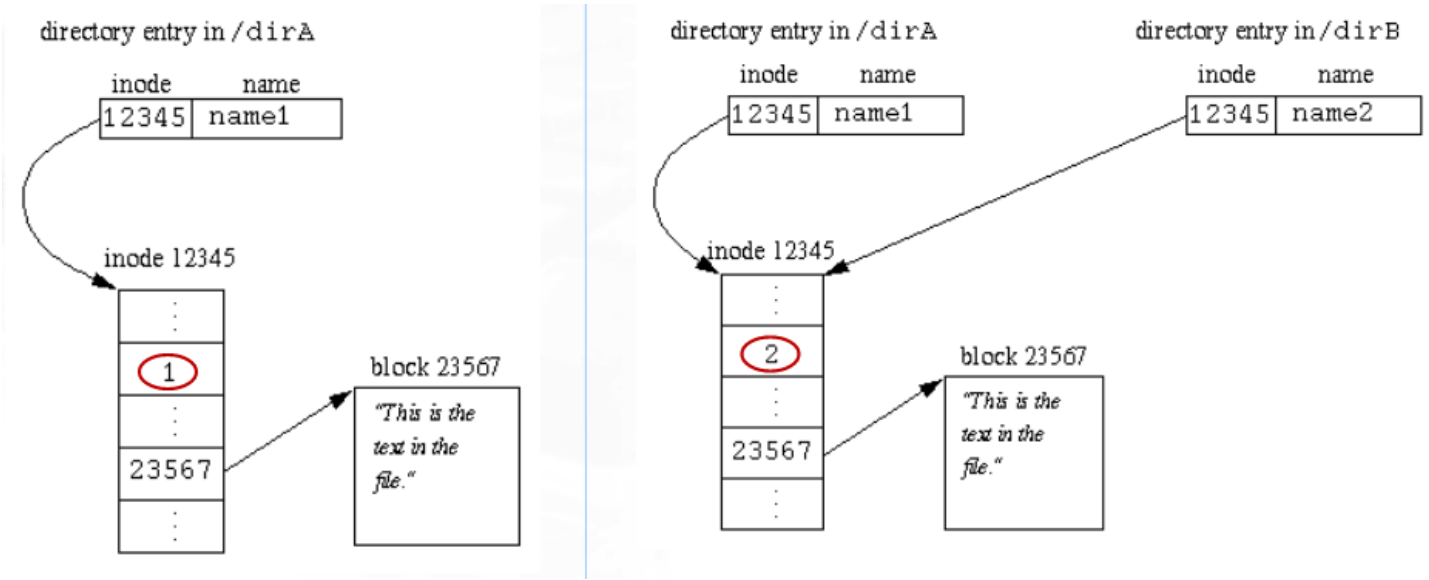

Figura 4: Diagrama ilustrando o que acontece após a criação de um *hard link* `/dirB/name2`

Para criar um hard link, podemos usar o seguinte comando no terminal:

```
$ ln path1_alvo_do_Link  path2_nome_do_arquivo_link
```

Por exemplo, na Figura 4, o arquivo `/dirB/name2` foi criado como um *hard link* do arquivo alvo `/dirA/name1`

```
$ ln /dirA/name1 /dirB/name2
```

Observem na figura que nas entradas do diretório de name1 e name2 temos o mesmo número de inode. Mas que houve uma mudança dentro do inode… 1 -> 2 (circulado em vermelho na Figura 4). Essa informação no inode mostra que agora temos 2 entradas de diretório apontando para o mesmo inode.

Também existe uma chamada de sistema que os programadores podem utilizar:

```
int link("/dirA/name1","/dirB/name2")
    // Cria um link simbólico (path2 -> path1)
```

TAREFA

13) No terminal, crie um arquivo `teste3` (com algum conteúdo texto dentro) e crie em seguida um *hard link* com o nome `teste4` usando **ln .** Depois verifique o resultado usando

```
$ stat teste3
$ stat teste4
```

Temos o mesmo inode? Perceba que para o sistema operacional, depois que um hard link é criado, não há diferença nenhuma entre ele e o arquivo alvo… o que muda é apenas o número de links marcado no próprio inode do arquivo.

14) Agora apague o arquivo que você havia criado, fazendo:

```
$ rm teste3
```

O conteúdo do arquivo foi realmente apagado? Faça um `cat teste4` para testar…

15) Agora crie um diretório dirA (mkdir dirA). E agora tente criar um *hard link* para ele:

```
$  ln dirA meulink
```

O que aconteceu? Você saberia dizer o porquê? **[RESPONDA NO FORMULÁRIO ONLINE]**

Montando diretórios/partições (EXTRA)

Para quem tiver interesse, no link a seguir vocês encontram uma aula bem clara sobre como funcionam partições e montagem de diretórios/partições:

- Curso GNU Linux - Aula 17 - Trabalhando com sistemas de arquivos
    - Obs: nessa aula, é utilizada uma VM para se trabalhar com partições de modo virtual. Caso você tenha interesse em usar VM, seguem alguns links sobre o VirtualBox:
        - Virtualbox – O Que É? Porque Utilizar Para Criar Seu Ambiente De Maquina Virtual?
        - Configuração de uma máquina virtual VirtualBox com sistema operacional Linux Fedora (Instalação no Windows)