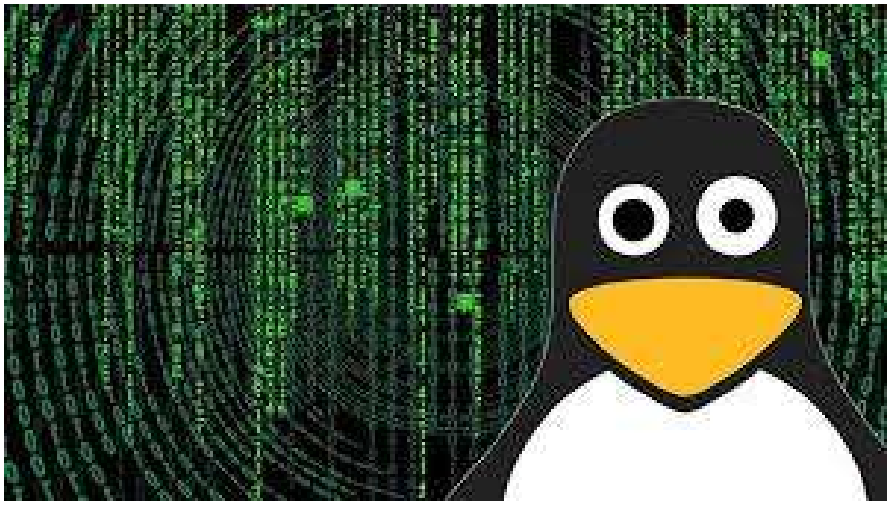
ESCOLA SUPERIOR DE TECNOLOGIA

Sistemas Operativos

Licenciatura em Engenharia de Sistemas Informáticos

Trabalho Prático



Luís Gonçalves 18851 – a18851@alunos.ipca.pt

Pedro Costa 13747 – a13747@alunos.ipca.pt

Arnaldo Silva 14289 – a14289@alunos.ipca.pt

Gonçalo dos Santos 11359 – a11359@alunos.ipca.pt

Índice

I

Implementação um conjunto de comandos para manipulação de ficheiros

Introdução

O objetivo é deste programa será implementar sete comandos distintos, que permitam a visualização, cópia, concatenação, contagem, exclusão, informações e listagem de arquivos e diretórios. Cada comando deve apresentar mensagens de erro adequadas em caso de falha na execução. O desafio requer habilidades de programação em C, bem como conhecimentos em funções de chamada ao sistema.

a) Mostrar ficheiro

A função "mostraFicheiro" implementa o comando "mostra ficheiro", que tem como objetivo exibir todo o conteúdo de um arquivo na saída padrão (stdout). Para isso, são utilizadas as seguintes funções de chamada ao sistema:

open: essa função abre um arquivo especificado por seu nome e retorna um descritor de arquivo (fd). No comando "mostra ficheiro", o arquivo é aberto apenas para leitura, com a flag O_RDONLY. A função retorna -1 em caso de erro na abertura do arquivo.

read: essa função lê um número especificado de bytes de um arquivo aberto e armazena no buffer fornecido como parâmetro. No comando "mostra ficheiro", é usada para ler o conteúdo completo do arquivo, um buffer por vez, até o final do arquivo. Ela retorna o número de bytes lidos ou -1 em caso de erro.

write: essa função escreve um número especificado de bytes em um arquivo aberto. No comando "mostra ficheiro", é usada para escrever os bytes lidos do arquivo no stdout (saída padrão). Ela retorna o número de bytes escritos ou -1 em caso de erro.

close: essa função fecha um arquivo aberto especificado pelo descritor de arquivo (fd). No comando "mostra ficheiro", é usada para fechar o arquivo aberto anteriormente. Ela retorna -1 em caso de erro no fechamento do arquivo.

Além disso, são utilizadas constantes definidas no código para determinar o tamanho do buffer (BUFFER_SIZE) usado para ler e escrever o conteúdo do arquivo. A função também exibe mensagens de erro apropriadas usando a função perror quando ocorrem erros nas chamadas de sistema, seguida da função exit para terminar o programa com falha.

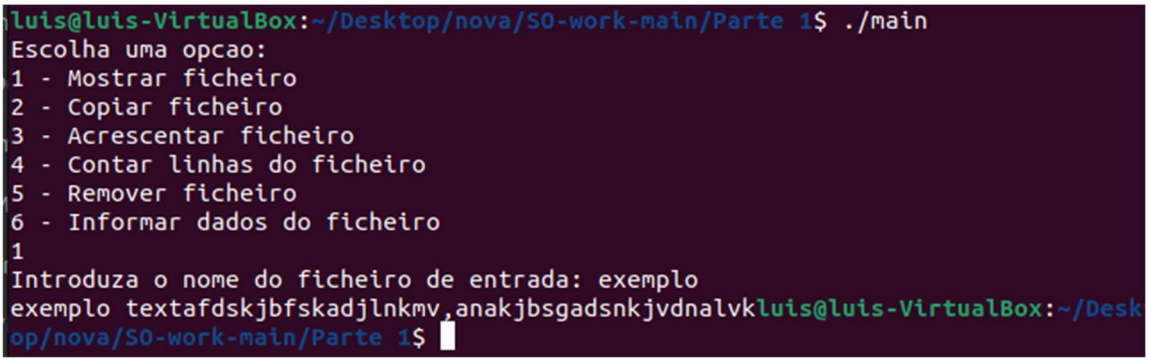


Figura 1 - Exemplo de como usar o 'mostrar ficheiro'

b) Copiar ficheiro

A função "copiaFicheiro" é usada para copiar um arquivo especificado pelo caminho "ficheiro" para um novo arquivo com o mesmo conteúdo, mas com um nome diferente (adicionando a extensão “. copia").

As chamadas do sistema utilizadas nesta função são:

- **open (ficheiro, O_RDONLY):** abre o arquivo especificado pelo caminho "ficheiro" em modo só leitura. Se o arquivo não puder ser aberto, a função retorna -1 e o erro é tratado com a chamada perror e a saída do programa com exit.

- **snprintf(nomeCopia, PATH_MAX, "%s.copia", ficheiro):** gera o nome do arquivo de cópia, que será o nome do arquivo original com a extensão ".copia". A função snprintf escreve o resultado em nomeCopia, que é um array de caracteres especificado com tamanho máximo PATH_MAX.

- **open (nomeCopia, O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH):** abre o arquivo de cópia em modo somente escrita (O_WRONLY), cria o arquivo se ele não existir (O_CREAT) e trunca o arquivo para tamanho zero se ele já existir (O_TRUNC). Os modos de acesso especificados com S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH permitem que o arquivo possa ser lido e gravado pelo usuário, grupo e outros usuários. Se a função open falhar, o erro é tratado com a chamada perror e a saída do programa com exit.

- **read(fdIn, buffer, BUFFER_SIZE):** lê dados do arquivo original usando o descritor de arquivo fdIn e armazena os dados lidos no buffer especificado buffer. O tamanho máximo de bytes a serem lidos é especificado por BUFFER_SIZE, que é uma constante pré-definida. A função retorna o número de bytes lidos, que pode ser menor que BUFFER_SIZE se o final do arquivo for alcançado.

- **write(fdOut, buffer, bytesPorLeitura):** escreve os dados lidos do arquivo original no arquivo de cópia usando o descritor de arquivo fdOut. A função retorna o número de bytes escritos. Se o número de bytes escritos for diferente do número de bytes lidos, ocorreu um erro e o programa é interrompido.

- **close(fdIn):** fecha o descritor de arquivo do arquivo original fdIn. Se a função close falhar, o erro é tratado com a chamada perror e a saída do programa com exit.

close(fdOut): fecha o descritor de arquivo do arquivo de cópia fdOut. Se a função close falhar, o erro é tratado com a chamada perror e a saída do programa com exit.

```
Escolha uma opcao:
1 - Mostrar ficheiro
2 - Copiar ficheiro
3 - Acrescentar ficheiro
4 - Contar linhas do ficheiro
5 - Remover ficheiro
6 - Informar dados do ficheiro
2
Introduza o nome do ficheiro de entrada: exemplo
O ficheiro exemplo foi copiado para exemplo.copia
```

Figura 2 - teste-exemplo do 'copiar ficheiro'

c) Acrescentar ficheiro

A função "acrescentaFicheiro" copia o conteúdo de um ficheiro de origem para um ficheiro de destino, acrescentando-o ao final do ficheiro de destino.

Para realizar esta operação, a função utiliza as seguintes chamadas do sistema:

- **int open(const char *path, int flags) -** Abre um ficheiro. É usada para abrir o ficheiro de origem e o ficheiro de destino. O primeiro argumento é uma string que representa o caminho para o ficheiro, e o segundo argumento são as flags que indicam o modo de abertura do ficheiro (O_RDONLY para leitura e O_WRONLY | O_APPEND para escrita com a flag O_APPEND para acrescentar ao final do ficheiro).
- **ssize_t read(int fd, void *buf, size_t count) -** Lê dados de um ficheiro. É usada para ler o conteúdo do ficheiro de origem. O primeiro argumento é o descritor de ficheiro do ficheiro de origem, o segundo argumento é um buffer onde os dados lidos serão armazenados, e o terceiro argumento é o número máximo de bytes a serem lidos.
- **ssize_t write(int fd, const void *buf, size_t count)** - Escreve dados num ficheiro. É usada para escrever o conteúdo lido do ficheiro de origem para o ficheiro de destino. O primeiro argumento é o descritor de ficheiro do ficheiro de destino, o segundo argumento é um buffer que contém os dados a serem escritos, e o terceiro argumento é o número de bytes a serem escritos.
- **int close(int fd) -** Fecha um descritor de ficheiro. É usada para fechar os descritores dos ficheiros de origem e destino após ter sido realizada a operação de cópia.

```
Escolha uma opcao:
1 - Mostrar ficheiro
2 - Copiar ficheiro
3 - Acrescentar ficheiro
4 - Contar linhas do ficheiro
5 - Remover ficheiro
6 - Informar dados do ficheiro
3
Introduza o nome do ficheiro de entrada: exemplo
Introduza o nome do ficheiro de saida: exemplo.copia
O ficheiro exemplo foi acrescentado ao ficheiro exemplo.copia
```

Figura 3 - Como copiar o ficheiro

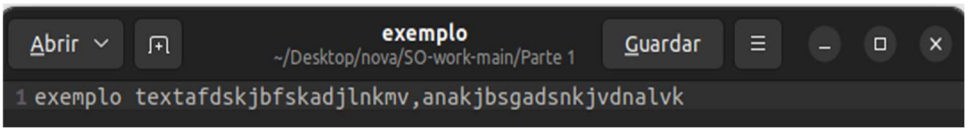


Figura 4 - exemplo.txt

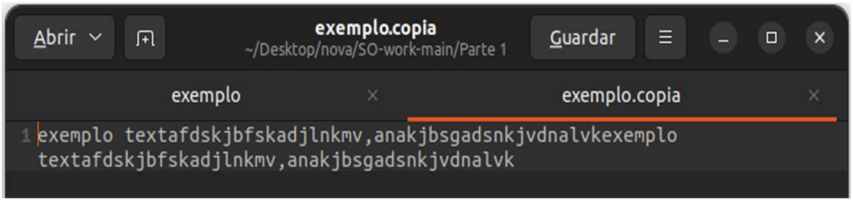


Figura 5 - exemplo.copia.txt

d) Contar ficheiro

A função "contaFicheiro" utiliza as seguintes funções de chamada do sistema:

- **open ():** É utilizada para abrir o ficheiro especificado em modo de leitura (O_RDONLY). Se o ficheiro não puder ser aberto, a função retorna -1 e uma mensagem de erro é impressa no terminal através da função perror().
- **read():** É utilizada para ler o conteúdo do ficheiro em blocos de tamanho BUFFER_SIZE. A função retorna o número de bytes lidos e armazena os dados lidos no buffer fornecido. O valor de retorno é verificado para saber se ocorreu um erro na leitura.
- **close():** É utilizada para fechar o descritor de ficheiro após ter sido feita a leitura. A função retorna -1 em caso de erro ao fechar o ficheiro.

A função percorre cada bloco lido e conta o número de linhas do ficheiro, somando um a cada vez que um caractere de nova linha é encontrado no buffer. O número total de linhas é retornado como resultado da função. Se ocorrer algum erro durante a execução, a função imprime uma mensagem de erro no terminal usando a função perror() e sai do programa com um código de erro.

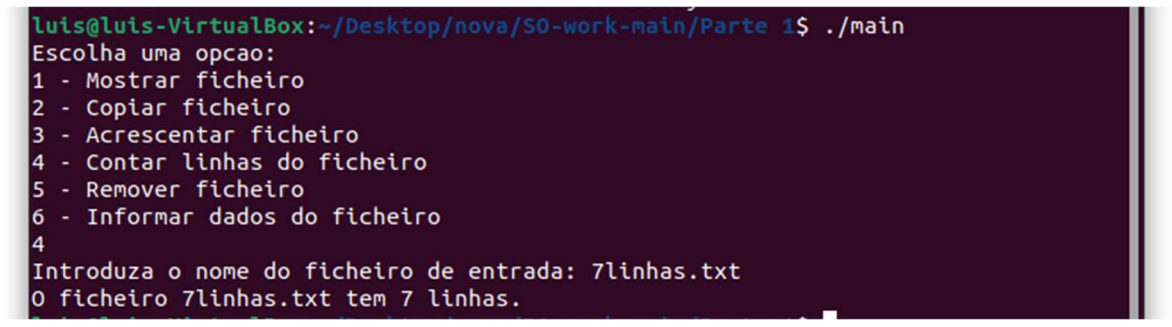


Figura 6 - Exemplo do contador de linhas

e) Apagar ficheiro

A função "removeFicheiro" utiliza a seguinte função de chamada do sistema:

- unlink(): É utilizada para remover o ficheiro especificado. A função retorna 0 se o ficheiro foi removido com sucesso ou -1 em caso de erro. O valor retornado é atribuído à variável "status".

```
luis@luis-VirtualBox:~/Desktop/nova/SO-work-main/Parte 1$ ls
a  exemplo  exemplo.copia  funcoes.c  funcoes.h  main  main.c
luis@luis-VirtualBox:~/Desktop/nova/SO-work-main/Parte 1$ ./main
Escolha uma opcao:
1 - Mostrar ficheiro
2 - Copiar ficheiro
3 - Acrescentar ficheiro
4 - Contar linhas do ficheiro
5 - Remover ficheiro
6 - Informar dados do ficheiro
7 - Listar diretoria
5
Introduza o nome do ficheiro de entrada: exemplo.copia
luis@luis-VirtualBox:~/Desktop/nova/SO-work-main/Parte 1$ ls
a  exemplo  funcoes.c  funcoes.h  main  main.c
```

Figura 7 - Exemplo da função removeFicheiro

f) Informar ficheiro

A função "informaFicheiro" utiliza as seguintes funções de chamada do sistema:

- **stat():** A função "stat" é utilizada para obter informações sobre o arquivo com o nome especificado no argumento "filename". A struct "file_stat" é preenchida com informações sobre o arquivo, incluindo o tipo de arquivo, as permissões de acesso, o proprietário do arquivo, o tamanho do arquivo, as datas e horários de criação, modificação e acesso, etc.

- **getpwuid():** A função "getpwuid" é utilizada para obter informações sobre o proprietário do arquivo, com base no ID do usuário especificado em "file_stat.st_uid". Essas informações incluem o nome do usuário, ID do usuário, diretório inicial do usuário, etc.

- **localtime():** A função "localtime" é utilizada para converter a data e hora armazenadas em "file_stat.st_ctime", "file_stat.st_mtime" e "file_stat.st_atime" em uma estrutura tm representando a hora local.

- **strftime():** A função "strftime" é utilizada para formatar a data e hora em uma string legível por humanos, com base em um formato de string especificado.

- **printf():** A função "printf" é utilizada para imprimir as informações obtidas a partir de "file_stat" e "pw" no formato especificado, utilizando especificadores de formato para inserir as informações corretas nas strings de saída.

```
luis@luis-VirtualBox:~/Desktop/nova/SO-work-main/Parte 1$ ./main
Escolha uma opcao:
1 - Mostrar ficheiro
2 - Copiar ficheiro
3 - Acrescentar ficheiro
4 - Contar linhas do ficheiro
5 - Remover ficheiro
6 - Informar dados do ficheiro
6
Introduza o nome do ficheiro de entrada: exemplo
exemplo:
Tipo: Arquivo regular
Número I-node: 283359
Dono: luis
Criado em: May 14 2023 21:43:
Última modificação: May 14 2023 17:43:
Último acesso: May 14 2023 21:45:{◆
luis@luis-VirtualBox:~/Desktop/nova/SO-work-main/Parte 1$
```

Figura 8 - Usar informar ficheiro

g) Lista [diretoria]

A função lista tem como objetivo listar os arquivos e diretórios contidos em um diretório especificado. Ela faz uso das seguintes chamadas do sistema:

- **opendir:** abre um diretório especificado, retornando um ponteiro para a estrutura DIR.
- **readdir**: lê a próxima entrada do diretório aberto pelo opendir, retornando um ponteiro para a estrutura dirent correspondente.
- **stat:** obtém informações sobre o arquivo/diretório especificado, armazenando-as na estrutura stat.
- **closedir**: fecha o diretório aberto pelo opendir.

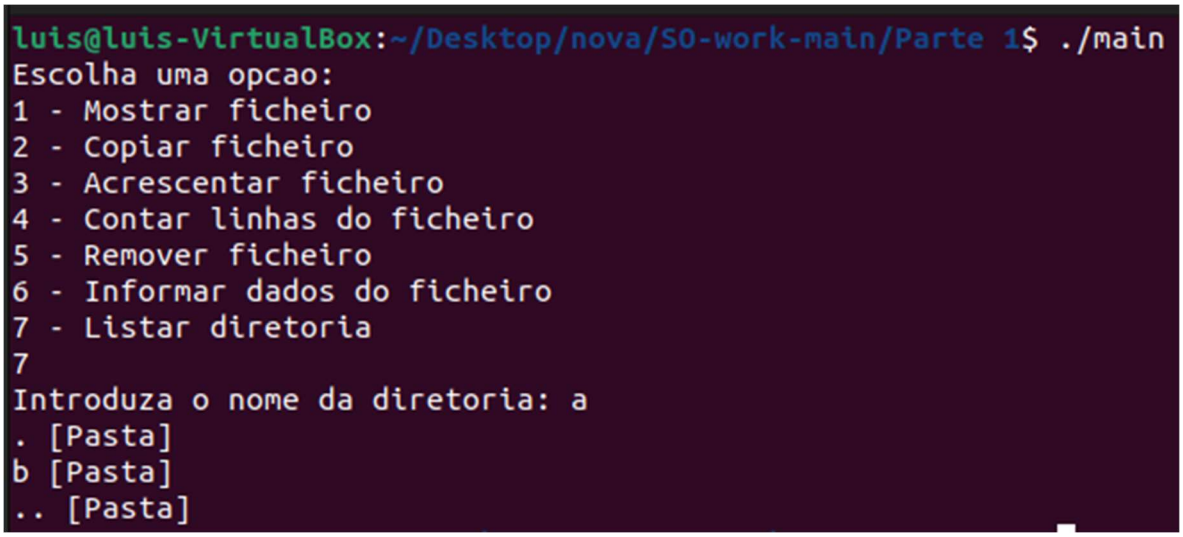


Figura 9 - Como listar diretorias

II

Implementação de um interpretador de linha de comandos

Introdução

Este relatório apresenta a implementação de um interpretador de comandos personalizado em linguagem C para sistemas operativos Linux. O interpretador foi projetado para ler uma sequência de caracteres do utilizador através do terminal, interpretar essa sequência como um comando e os respetivos argumentos, e executá-los no sistema.

O programa foi projetado com uma interface de utilizador simples e intuitiva. Ao ser iniciado, apresenta o símbolo "%" para indicar que está pronto para ler um novo comando. O utilizador pode então inserir comandos, que são executados num novo processo. O interpretador espera até que o comando esteja concluído e, em seguida, informa o utilizador se o comando foi concluído com sucesso ou não, através do código de erro/terminação.

Foram implementadas quatro funções principais, nomeadamente "criar", "apagar", "editar" e "listar". Cada uma destas funções é mapeada para um comando equivalente do Linux. A execução do interpretador pode ser encerrada pelo utilizador através do comando "termina".

Desenvolvimento e Funcionamento do Código

O código começa por definir algumas constantes para o comprimento máximo do comando e o número máximo de argumentos. Em seguida, entra num ciclo infinito onde lê o comando do utilizador, divide-o em argumentos e cria um novo processo para executar o comando.

O comando é lido do terminal usando a função fgets(). Em seguida, é dividido em argumentos usando a função strtok(), que divide a string em tokens com base nos espaços.

Se o comando inserido for "termina", o interpretador sai do ciclo e termina a execução. Caso contrário, cria um novo processo usando a função fork(). O processo filho substitui o comando "criar", "apagar", "editar" e "listar" pelos comandos equivalentes do Linux e executa o comando usando a função execvp()

Por fim, o processo pai espera que o processo filho termine e informa o utilizador do código de terminação do comando.

Conclusão

O interpretador de comandos personalizado apresentado neste relatório demonstra como é possível criar um programa em linguagem C que executa comandos do sistema operativo Linux. O programa faz uso eficiente das primitivas de execução genérica de processos disponíveis em C e fornece uma interface de utilizador simples e intuitiva.

Os resultados mostram que o interpretador é capaz de executar comandos e informar o utilizador sobre o seu sucesso ou falha.

III

Gestão de Sistemas de Ficheiros

Introdução

A gestão de sistemas de ficheiros em servidores virtuais é um processo de extrema importância para garantir a organização e segurança dos dados armazenados em discos virtuais. A criação de partições num disco virtual permite que diferentes sistemas de ficheiros sejam usados para armazenar dados da forma mais eficiente. Aliás, o uso do **Logical Volume Manager**, doravante tratado por **LVM,** permite que volumes físicos sejam combinados num único volume lógico, tornando a gestão de discos virtuais mais flexível e escalável.

A montagem de sistemas de ficheiros em diretórios específicos permite que os dados sejam acessados pelos utilizadores com maior facilidade e segurança. É importante configurar as permissões corretas para garantir que apenas os utilizadores autorizados possam aceder e alterar os dados.

Em suma, a gestão de sistemas de ficheiros nos servidores virtuais envolve diferentes etapas, como a criação de partições, o uso de LVM e a escolha do sistema de ficheiros correto.

Mas afinal, o que é o LVM?

O LVM é o sistema de gestão de ficheiros usado pelo sistema operativo Linux.

Com o LVM, é possível criar volumes lógicos que se ajustam às necessidades específicas de uma aplicação ou utilizador, independentemente do tamanho ou localização dos discos físicos. O LVM permite também expansão e redução de volumes em tempo real, redimensionamento dinâmico de volumes, entre outras coisas…

A diferença do LVM e os sistemas de partições tradicionais é que estas eram fixas e redimensioná-las nem sempre é um processo rápido.

Como funciona o LVM?

1. discos físicos são particionados em áreas chamadas de volumes físicos (PVs).
2. volumes físicos são combinados em grupos de volumes (VGs).
3. os volumes lógicos (LVs) são criados nos grupos de volumes e formatados com sistemas de ficheiros.

Arquitetura da LVM

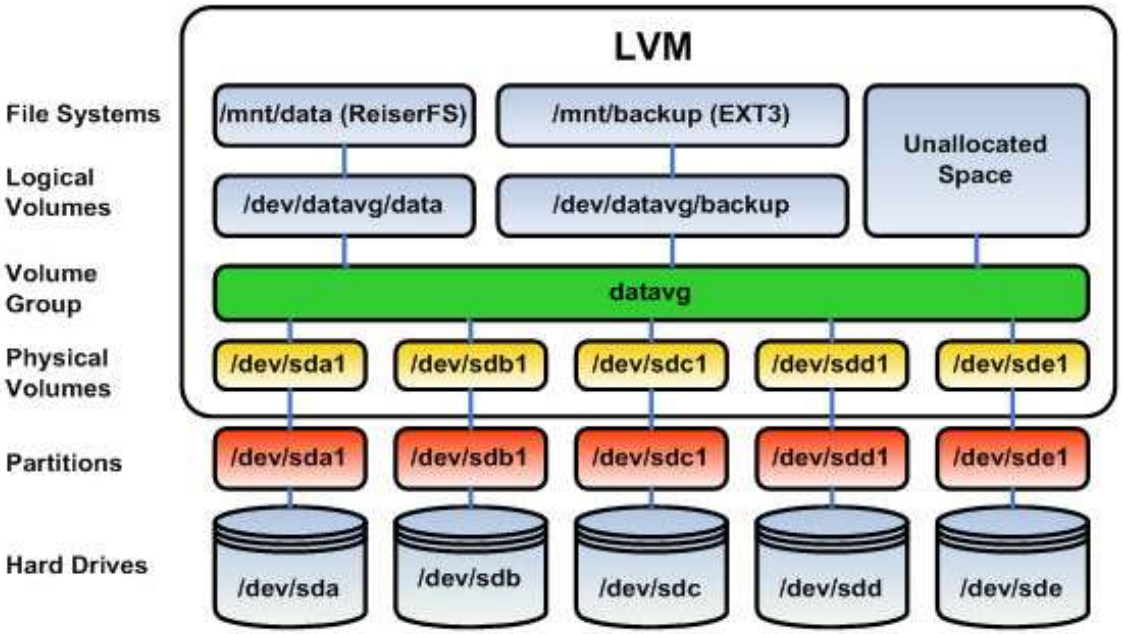
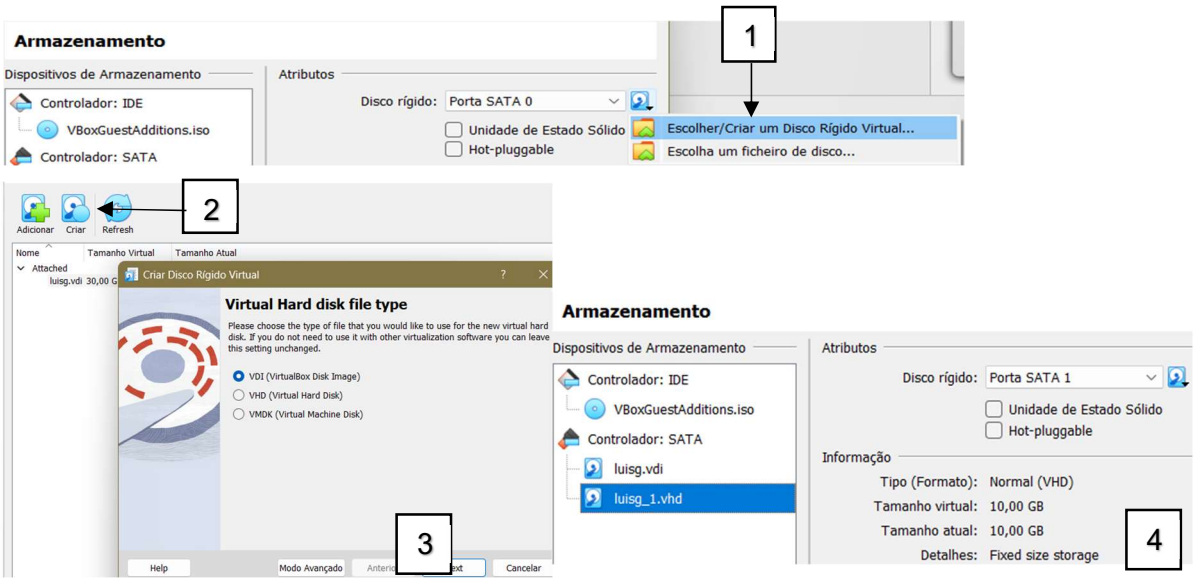


Figura 10 - Esquema arquitetónico do gestor de Volumes Lógicos

Como podemos ver na figura, que deve ser analisada no sentido ascendente temos na base os discos. A partir desses discos temos as partições. **Até aqui é usado o método tradicional**. Depois é aplicado LVM. A cada partição corresponde um volume físico. Depois vêm **os grupos de volumes que podem agrupar um ou vários volumes físicos**. E depois os grupos de volumes dão origem aos volumes lógicos que podem ser criados a partir de um grupo ou pode ser particionado um grupo em vários volumes lógicos.

a) Criação do disco e criação da partição



Como podemos ver em cima o disco foi criado com sucesso.

Instalei o LVM através do comando:

```
sudo apt-get install lvm2
```

Podemos obter as informações sobre as partições no disco rígido do sistema através de:

```
sudo fdisk -l
```

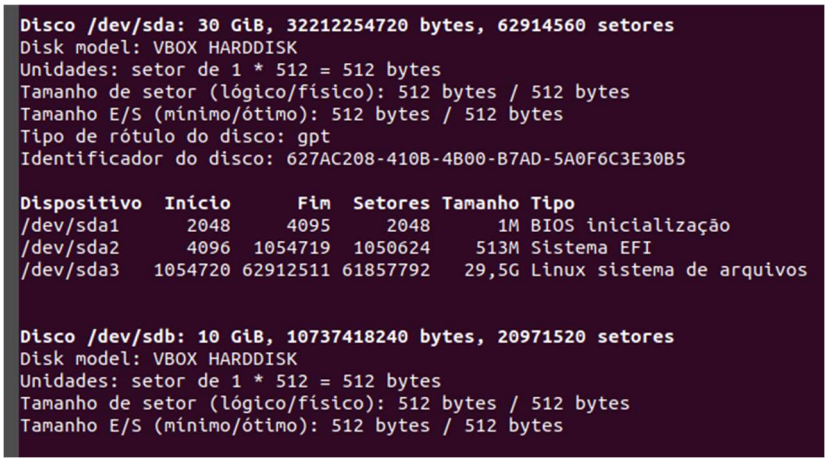


Figura 11- Lista de informações sobre as partições no disco

E criamos uma partição através do comando:

```
sudo fdisk [diretório do disco]
```

Como podemos ver na figura abaixo.

Não é possível criar uma partição com exatamente o mesmo tamanho do disco (10GB) porque:

1. Há espaço em disco reservado para a metadata
2. A forma como os fabricantes de discos rígidos definem um gigabyte. Normalmente, os fabricantes definem um gigabyte como 1.000.000.000 bytes, enquanto os sistemas operativos definem um gigabyte como 1.073.741.824 bytes.

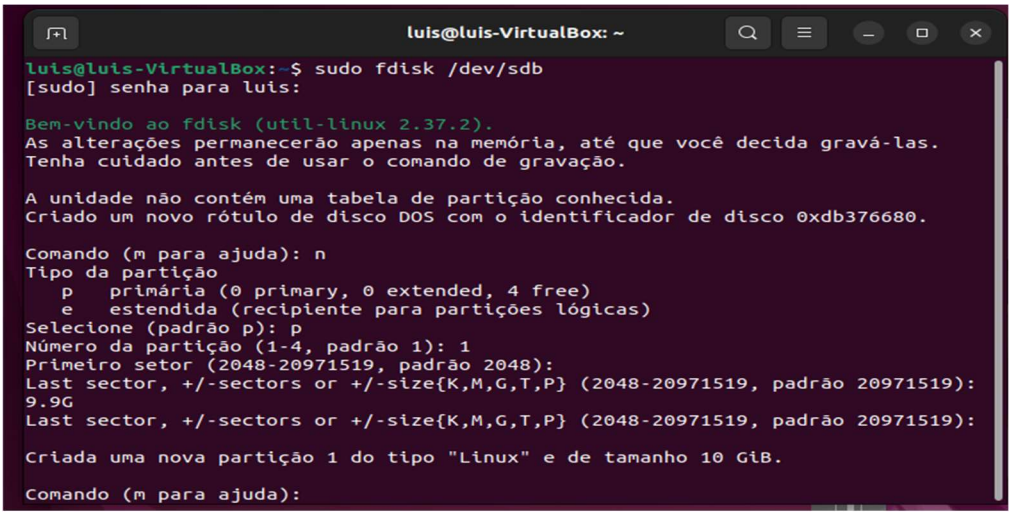
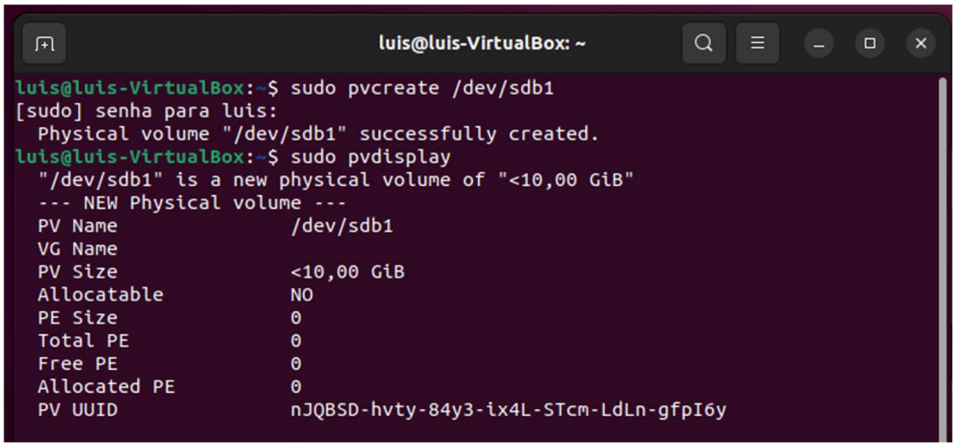


Figura 12 - Criação da partição no disco sdb

b) Criação de Volumes físicos e lógicos



Criação do volume físico com o comando:

`sudo pvcreate [diretório da partição]`

e do comando:

`sudo pvdisplay`

para que seja apresentado no terminal todas as informações sobre o volume físico.

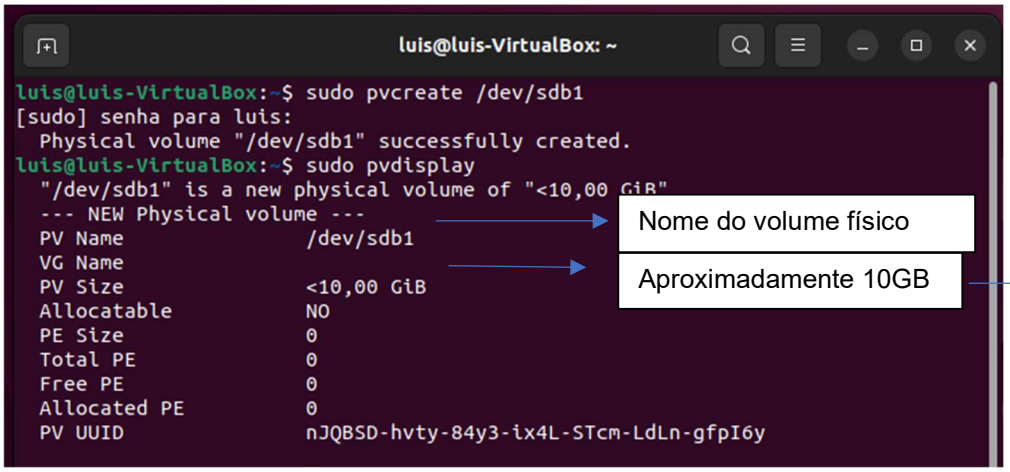


Figura 13 - Criação do volume físico

Porquê?

Não é possível criar uma partição com exatamente o mesmo tamanho do disco (10GB) porque:

1. Há espaço em disco reservado para a metadata
2. A forma como os fabricantes de discos rígidos definem um gigabyte. Normalmente, os fabricantes definem um gigabyte como 1.000.000.000 bytes, enquanto os sistemas operativos definem um gigabyte como 1.073.741.824 bytes.

```
luis@luis-VirtualBox:~$ sudo lvcreate -L 4.9G -n vol01
  No command with matching syntax recognised.  Run 'lvcreate --help' for more information.
  Nearest similar command has syntax:
  lvcreate --type error -L|--size Size[m|UNIT] VG
  Create an LV that returns errors when used.
```

Aqui tentamos criar o volume lógico, sem sucesso, pois não tinhamos criado o grupo de volumes. Ver figura 1.

```
luis@luis-VirtualBox:~$ sudo vgcreate grupo /dev/sdb1
  Volume group "grupo" successfully created
luis@luis-VirtualBox:~$ sudo lvcreate -L 5GB -n volume1 grupo
  Logical volume "volume1" created.
luis@luis-VirtualBox:~$ sudo lvcreate -L 5GB -n volume2 grupo
  Volume group "grupo" has insufficient free space (1279 extents): 1280 required.
luis@luis-VirtualBox:~$ sudo lvremove /dev/sdb1/grupo/volume1
  "sdb1/grupo/volume1": Invalid path for Logical Volume.
luis@luis-VirtualBox:~$ sudo lvremove /dev/sdb1/grupo/volume1
  "sdb1/grupo/volume1": Invalid path for Logical Volume.
luis@luis-VirtualBox:~$ sudo lvremove /dev/grupo/volume1
Do you really want to remove and DISCARD active logical volume grupo/volume1? [y/n]: Y
  Logical volume "volume1" successfully removed
luis@luis-VirtualBox:~$ sudo lvcreate -L 4.9GB -n volume1 grupo
  Rounding up size to full physical extent 4,90 GiB
  Logical volume "volume1" created.
luis@luis-VirtualBox:~$ sudo lvcreate -L 4.9GB -n volume2 grupo
  Rounding up size to full physical extent 4,90 GiB
  Logical volume "volume2" created.
```

Figura 14 - Criação de grupo de volumes e volumes lógicos

vgcreate [diretório do grupo] : cria o grupo de volumes

lvcreate -L [tamanho do volume lógico] -n [nome do volume] [nome do grupo de volumes]:

- cria o volume lógico
- -L para especificar tamanho do volume lógico
- -n para definir nome
- é imperativo ter o nome do grupo de volumes ao qual pertence

lvremove [caminho do volume lógico] para remover volume lógico

No fim podemos ver que os volumes lógicos denominados volume1 e volume2 criados com sucesso.

Podemos conferir os volumes lógicos através do lvdisplay (mais detalhado) ou do lvscan (mais básico).

c)

```
luis@luis-VirtualBox:~$ sudo mkfs.ext4 /dev/grupo/volume1
[sudo] senha para luis:
mke2fs 1.46.5 (30-Dec-2021)
A criar sistema de ficheiros com 1285120 4k blocos e 321280 inodes
UUID do sistema de ficheiros: ec5fb73f-f886-4f17-ad46-0f407fe6aa2d
Seguranças de super-blocos armazenadas em blocos:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

A alocar tabelas de grupo: feito
A escrever tabelas de inodes: feito
A criar diário (16384 blocos): feito
A escrever super-blocos e informação de contabilidade do sistema de ficheiros:  feito

luis@luis-VirtualBox:~$ sudo mkfs.ext3 /dev/grupo/volume2
mke2fs 1.46.5 (30-Dec-2021)
A criar sistema de ficheiros com 1285120 4k blocos e 321280 inodes
UUID do sistema de ficheiros: 50495d5a-3870-484b-bda4-ed85fc60d3b4
Seguranças de super-blocos armazenadas em blocos:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

A alocar tabelas de grupo: feito
A escrever tabelas de inodes: feito
A criar diário (16384 blocos): feito
A escrever super-blocos e informação de contabilidade do sistema de ficheiros:  feito
```

Figura 15 - Criação dos sistemas de ficheiros do tipo ext4 e ext3

O comando:

`sudo mkfs.ext4[caminho_do_volume_logico]`

para criar o sistema de ficheiros do tipo ext4 e o comando:

`sudo mkfs.ext3[caminho_do_volume_logico]`

para o tipo ext3.

Ext4 e Ext3

O ext4 e o ext3 são sistemas de fichieros usados em Linux. O ext3 é um sistema mais antigo, lançado em 2001, e é uma extensão do sistema de ficheiros ext2, que não suportava journaling[1].

Por outro lado, o ext4 foi lançado em 2008 e é uma evolução do ext3. Ele suporta tamanhos de ficheiros maiores, pode lidar com sistemas de até um exabyte e tem um melhor desempenho em relação ao ext3.

Ambos os sistemas são amplamente utilizados em sistemas operativos Linux. No entanto, o ext4 é considerado uma atualização mais avançada e é geralmente preferido em novas instalações.

1- O journaling é um recurso que ajuda a proteger os dados em caso de falhas do sistema ou desligamentos inesperados.

d)

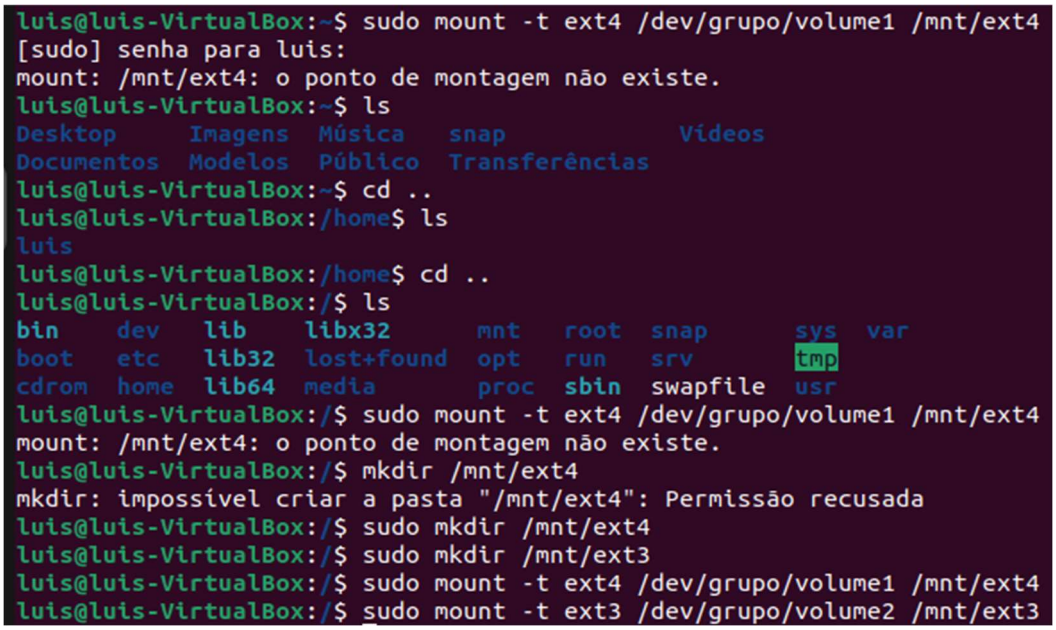


Figura 16 - Como montar os sistemas de ficheiros

Aqui montamos os sistemas de ficheiros. É importante realçar que devemos navegar para as diretorias corretas a fim de evitar os erros acima identificados.

Comando para montar:

`mount -t [tipo_sistema] [caminho_do_volume_logico] [caminho_do_sistema_ficheiros]`

-t: diz ao kernel para anexar o sistema de ficheiros encontrado no dispositivo (que é do tipo type) no diretório

e)

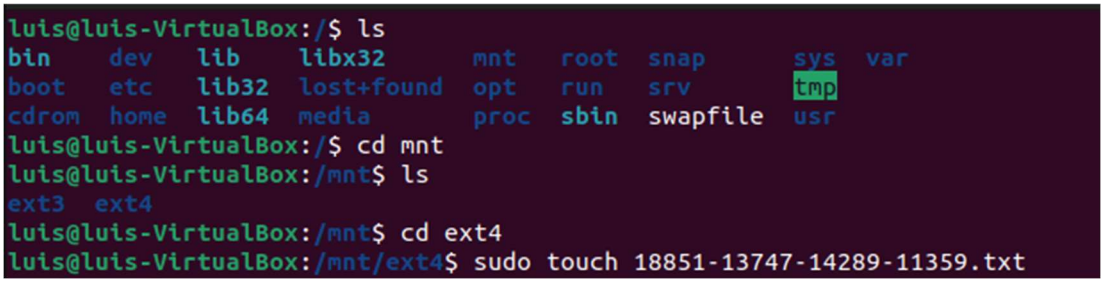
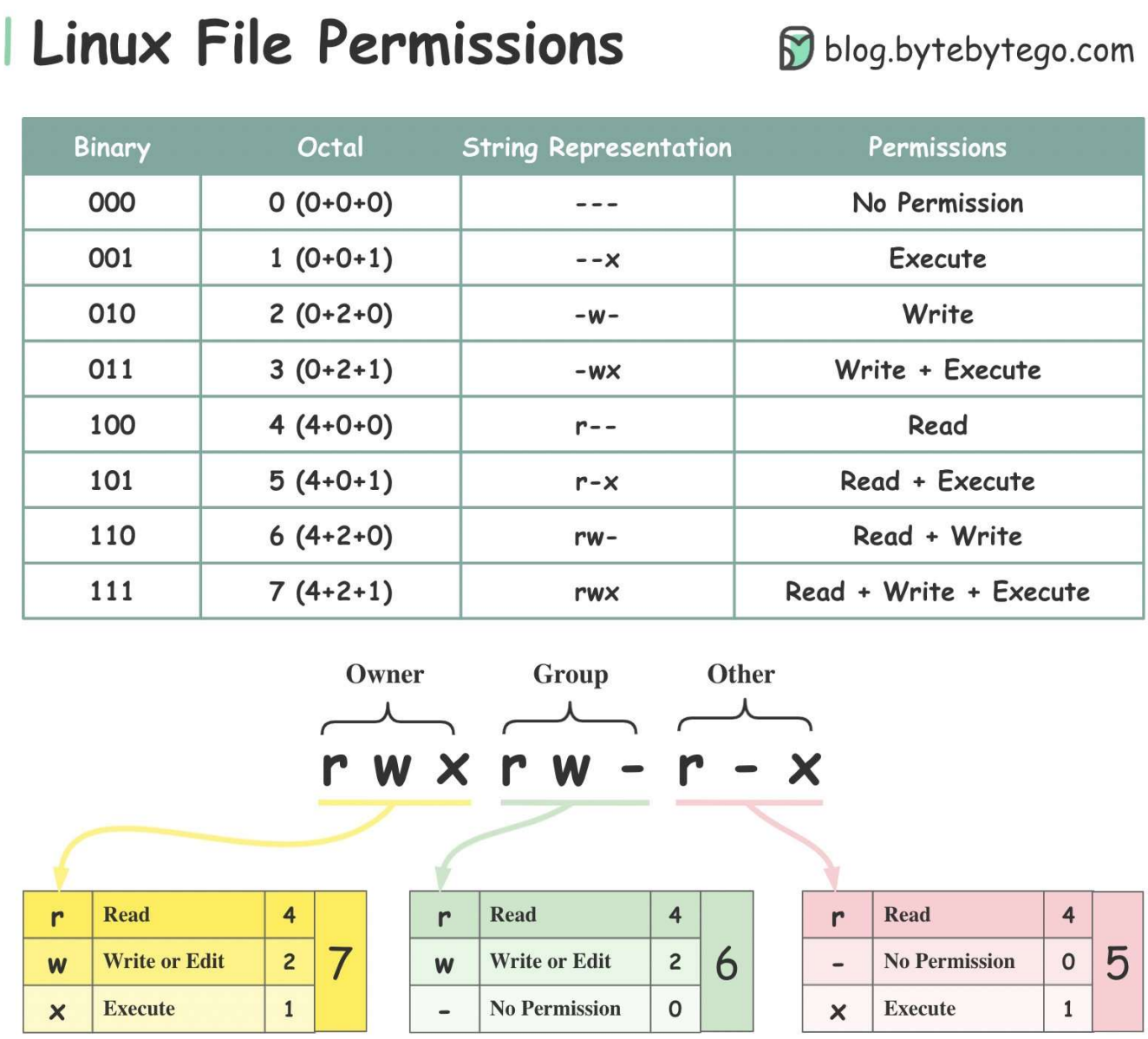


Figura 17 - Criação do ficheiro pedido na alínea e)

Permissões em linux



Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute

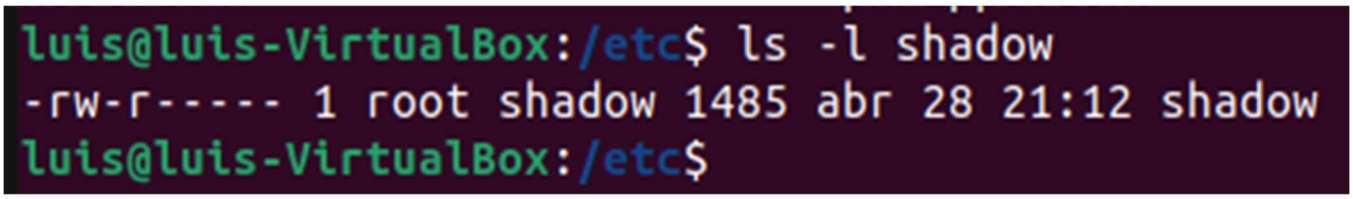
Figura 18 - Permissões de ficheiros no Linux

```
luis@luis-VirtualBox:/mnt/ext4$ sudo chmod 604 18851-13747-14289-11359.txt
```

Figura 19 - Como alteramos as permissões de um ficheiro

Usamos o comando chmod para definir as permissões. Cada algarismo corresponde a um tipo de utilizador (dono, grupo, outros) e como o dono tem permissão de escrita e leitura, ou seja, 6, grupo sem permissão, isto é, 0 e outros com permissões de leitura que resulta no 0 chegamos ao número 604 como podemos ver utilizado na figura acima.

f)



Usamos o comando ls –l [nome_do_ficheiro] ter acesso às informações do ficheiro incluindo permissões.