Instituto Politécnico do Cávado e do Ave
Escola Superior de Tecnologia

Manipulação de Ficheiros - Sistemas Operativos

Bruna Raquel Noversa Macieira – 21139
Ricardo Daniel da Silva Teixeira – 20080
Francisco Rafael Dias Pereira - 21156

VERSÃO FINAL

Relatório do Trabalho Prático realizado no âmbito da
Unidade Curricular de Sistemas Operativos do curso de
Licenciatura em Engenharia de Sistemas Informáticos

Prof. Fernando Gomes

Barcelos, 1 de Maio de 2022

Resumo

Este documento retrata o trabalho desenvolvido no âmbito da unidade curricular de Sistemas Operativos no curso de Licenciatura em Engenharia de Sistemas Informáticos, lecionado na Escola Superior de Tecnologia do Instituto Politécnico do Cávado e do Ave.

Resumidamente, o trabalho consiste em dois capítulos: no primeiro, será abordada a implementação de vários comandos para manipular ficheiros e diretorias de um sistema operativo; no segundo, será abordado o sistema de ficheiros de discos de armazenamento do sistema operativo.

Abstract

This document portrays the work developed in Operating Systems curricular unit in the bachelor's degree of Computer Systems Engineering, taught at the School of Technology in Polytechnic Institute of Cávado and Ave.
Briefly, the work consists of two chapters.
The first one will address the implementation of various commands to manipulate files and directories of an operating system.
And the second one will address the operating system's storage disks, volumes, partitions and the file system itself.

Índice

Lista de figuras

Abreviaturas e Símbolos

Lista de abreviaturas (ordenadas por ordem alfabética)

GB	*Gigabyte* (unidade de capacidade de armazenamento de dados)
GID	*Group Identifier from the Owner of the File*
UID	*User Identifier from the Owner of the File*
VDI	*VirtualBox Disk Image*

Capítulo 1 - Implementação de comandos para manipulação de ficheiros

Introdução

Neste primeiro capítulo, é abordada a implementação de comandos para manipulação de ficheiros e diretorias, sobretudo, no sistema operativo Linux.

Foi proposto pelo docente a implementação dos comandos:

- **mostra** <ficheiro> – Deve apresentar no ecrã todo o conteúdo do ficheiro indicado como parâmetro. Caso este não exista, o comando deve avisar o utilizador que o ficheiro não existe; (alínea resolvida pelo Ricardo – 20080)

- **copia** <ficheiro> – Este deve criar um novo ficheiro, cujo o nome será "<nomedoficheiro>.copia", cujo o conteúdo é uma cópia de todo o conteúdo do ficheiro passado como parâmetro no comando. Caso o ficheiro não exista, deve ser apresentado um aviso ao utilizador; (alínea resolvida pela Bruna – 21139)

- **acrescenta** <origem> <destino> – Este comando deve acrescentar todo o conteúdo do ficheiro de "origem" no final do ficheiro de "destino", ambos passados como parâmetro à frente do comando. Caso algum dos ficheiros não exista, este deve ser apresentado um aviso ao utilizador; (alínea resolvida pelo Francisco – 21156)

- **conta** <ficheiro> – Este comando deve contar o número de linhas existentes num ficheiro. Se o ficheiro não existir, deverá ser indicado ao utilizador uma mensagem de erro; (alínea resolvida pelo Ricardo – 20080)

- **apaga** <ficheiro> – Este comando deve apagar o ficheiro com o nome indicado. Se o ficheiro não existir, deve ser apresentado um aviso ao utilizador; (alínea resolvida pela Bruna – 21139)

- **informa** <ficheiro> – Este comando apresenta a informação do sistema de ficheiros em relação ao ficheiro indicado, nomeadamente, o tipo de ficheiro, i-node, o dono do ficheiro, e as datas de criação, leitura e modificação; (alínea resolvida pelo Francisco – 21156)

- **lista** <diretoria> – Este comando deve apresentar uma lista de todas as pastas e ficheiros existentes na diretoria indicada ou na diretoria atual se não especificada como

parâmetro. Adicionalmente, deve distinguir ficheiros simples de diretorias através de uma indicação textual. (alínea resolvida pelo Ricardo – 20080)

1.1 - Comando “mostra <ficheiro>”

Este comando foi implementado com a designação “_cat” e, assim que executado, mostra o nome do ficheiro na primeira linha e, de seguida, mostra todo o conteúdo do ficheiro.

Suporta vários ficheiros de uma execução só, ou seja, pode introduzir-se como parâmetro vários ficheiros que este executa todos sequencialmente.

Assim que ele não encontrar algum dos ficheiros, simplesmente apresenta a mensagem de erro para esse mesmo ficheiro, ou, caso não seja apresentado nenhum ficheiro como parâmetro, também devolverá uma mensagem de erro para o utilizador.

```
ricardoteixeira@MBPdeRicardo2 SO % ./_cat test.txt
test.txt:
jksenfkjesnfjk
esfsefsk
sefse
f
se
f
sefsefsegs

esgse
g
es
g
seg
es
g
se
g
```

Figura 1 - Execução do comando implementado "_cat".

```
ricardoteixeira@MBPdeRicardo2 SO % ./_cat
Error! No files given, please insert a filename!
ricardoteixeira@MBPdeRicardo2 SO % ./_cat naoexiste.txt
Error! File naoexiste.txt not found!
```

Figura 2 - Execução do comando para apresentar mensagens de erro.

Todo o código de implementação deste comando foi desenvolvido na linguagem de programação C, e encontra-se nos anexos deste documento com a designação “_cat.c”.

1.2 - Comando “copia <ficheiro>”

Este comando foi implementado com a designação “_cpy” e, assim que executado, mostra se os ficheiros foram copiados com sucesso ou se houve algum erro.

Apresenta uma mensagem de erro quando não é mencionado nenhum ficheiro como parâmetro na execução do comando ou caso o ficheiro não seja encontrado.

Este comando suporta vários ficheiros de uma execução só.

Como podemos verificar na Figura 3, executou-se o comando “ls” para verificar que não existem ficheiros “.copia” e, após executar o comando com dois ficheiros válidos como parâmetros, estes foram copiados e foram criadas cópias destes ficheiros com a extensão “.copia”, como podemos verificar ao executar pela segunda vez o comando “ls”.

De seguida, foi testado o comando com um ficheiro que não existe e este, como suposto, retornou uma mensagem de erro a avisar que o ficheiro não existe, assim como também avisou o utilizador que não foram introduzidos ficheiros como parâmetro quando testamos executar o comando por si só.

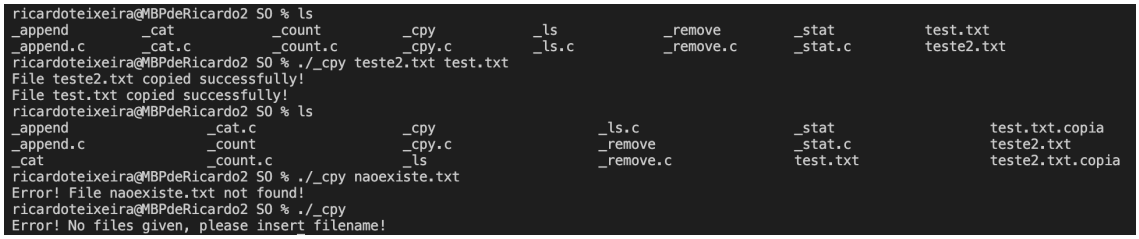

Figura 3 - Execução do comando "_cpy" com as várias possibilidades de retorno e execução do comando "ls" para verificar o funcionamento do "_cpy".

Todo o código de implementação deste comando foi desenvolvido na linguagem de programação C, e encontra-se nos anexos deste documento com a designação “_cpy.c”.

1.3 - Comando “acrescenta <origem> <destino>”

Este comando foi implementado com a designação “_append” e, assim que executado, mostra que o ficheiro foi modificado com sucesso. No entanto, quando não é dado um número de parâmetros par, ou seja, sempre um destino para uma origem, apresenta uma mensagem de erro ao utilizador.

Caso o ficheiro não exista, também apresenta um erro ao utilizador com o nome do ficheiro que não encontrou, e passa para o próximo par de ficheiros.

Como se pode verificar na Figura 4, ao executar o comando “cat” (comando responsável por apresentar o conteúdo de um ficheiro) ao ficheiro “test.txt”, pode perceber-se que apenas tem o nome “Bruna”, e ao executar “cat” aos três ficheiros “test.txt”, “teste2.txt” e “teste3.txt”, verifica-se que tem os nomes “Bruna”, “Ricardo” e “Francisco” respetivamente em cada um deles.

Ao executar o comando “_append” com os parâmetros de modo que o destino seja sempre o “test.txt” e as origens sejam “teste2.txt” e “teste3.txt”, este juntou os nomes “Ricardo” e “Francisco” ao ficheiro que continha o nome “Bruna”, ficando o ficheiro “test.txt” com os três nomes referentes aos três autores deste trabalho prático.

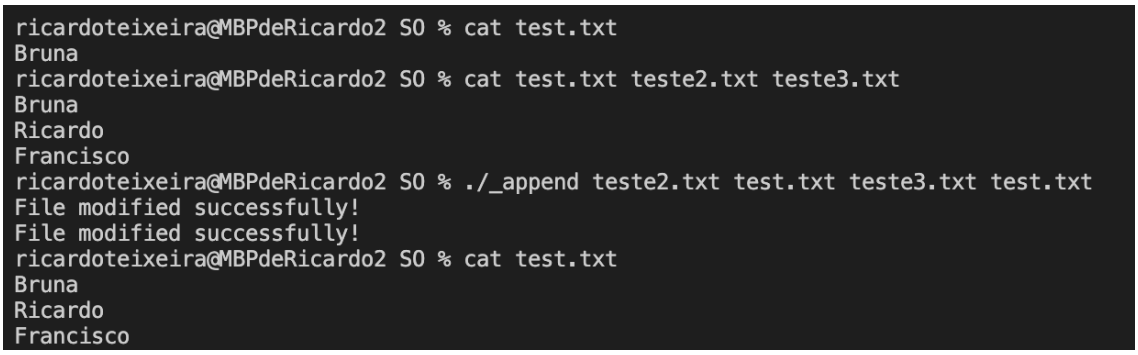

Figura 4 - Execução do comando "_append", bem como do comando "cat" para verificação.

Todo o código de implementação deste comando foi desenvolvido na linguagem de programação C, e encontra-se nos anexos deste documento com a designação "_append.c".

1.4 - Comando "conta <ficheiro>"

Este comando foi implementado com a designação "_conta" e, assim que executado, este verifica se há ficheiros nos parâmetros e se estes ficheiros são válidos ou não, caso contrário apresenta uma mensagem de erro.

De seguida, abre um ficheiro de cada vez, pela ordem dada nos parâmetros, e conta quantos "\n" tem, perfazendo assim o número de linhas nesse ficheiro e apresenta no ecrã para cada ficheiro o número de linhas no seu conteúdo. O "\n" é o caracter que define uma nova linha.

Na Figura 5, observa-se que, ao verificar o conteúdo do ficheiro "test.txt" com o comando "cat", pode contar-se 3 linhas de texto. E, ao executar o comando "_count" a esse mesmo ficheiro, pode verificar-se o previsto: 3 linhas.

Nos casos seguintes, colocou-se um ficheiro que não existe como parâmetro, também se executou o comando sem qualquer parâmetro e este apresentou as respetivas mensagens de erro.

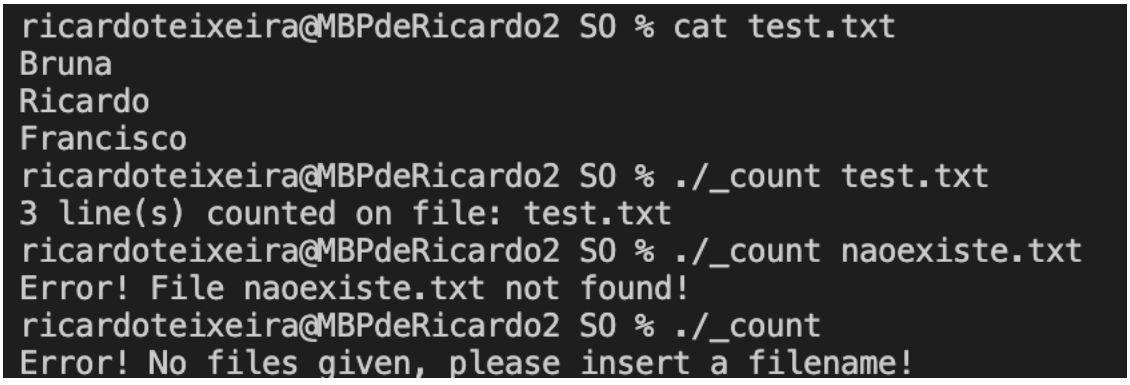

Figura 5 - Execução do comando "_count" em variadas situações e exeucação do comando "cat" para verificação.

Todo o código de implementação deste comando foi desenvolvido na linguagem de programação C, e encontra-se nos anexos deste documento com a designação "_count.c".

1.5 - Comando "apaga <ficheiro>"

Este comando foi implementado com a designação "_remove" e, assim que executado, este verifica se há ficheiros nos parâmetros, e se estes ficheiros são válidos ou não, caso contrário apresenta uma mensagem de erro.

De seguida, tenta aceder ao ficheiro, pela ordem dada nos parâmetros (suporta vários) e, caso seja possível aceder, este remove o ficheiro usando a função "unlink()" e apresenta no ecrã os ficheiros que eliminou com sucesso.

Na Figura 6, observa-se com o comando "ls" que os ficheiros "test.txt.copia" e "teste2.txt.copia" existem e, usando o comando "_remove", os dois ficheiros são removidos com sucesso, como se pode ver no final da Figura 6 ao executar o comando "ls" para listar os ficheiros da diretoria atual.

Nos casos seguintes, colocou-se um ficheiro que não existe como parâmetro, também se executou o comando sem qualquer parâmetro e este apresentou as respetivas mensagens de erro.

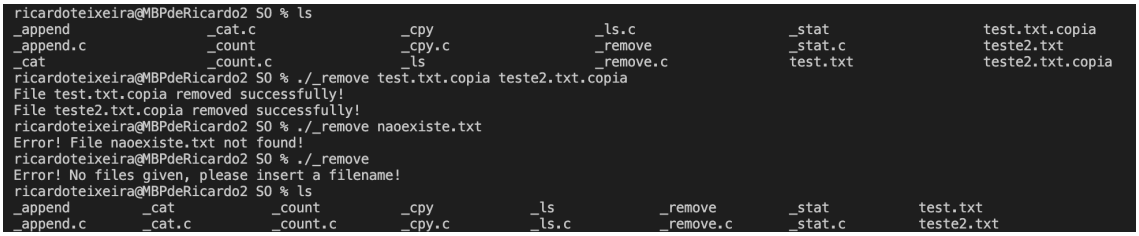

Figura 6 - Execução do comando "_remove" e execução do comando "ls" para verificação.

Todo o código de implementação deste comando foi desenvolvido na linguagem de programação C, e encontra-se nos anexos deste documento com a designação "_remove.c".

1.6 - Comando "informa <ficheiro>"

Este comando foi implementado com a designação "_stat" e, assim que executado, este verifica se há ficheiros nos parâmetros, e se estes ficheiros são válidos ou não, caso contrário apresenta uma mensagem de erro.

De seguida, tenta aceder ao ficheiro, pela ordem dada nos parâmetros (pois suporta vários) e, caso seja possível aceder, este apresenta no ecrã informações acerca do ficheiro, tais como: o tipo de ficheiro, i-node, informação sobre o utilizador que o criou,

desde o seu *username* em modo textual, como o UID, e GID, e por fim, as datas de criação, acesso e modificação do ficheiro.

Na Figura 7, pode observar-se um exemplo de execução deste comando, com o ficheiro "text.txt", e verificar que devolve os dados supracitados acerca do ficheiro.

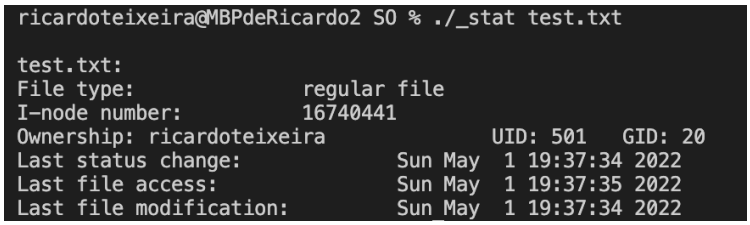

Figura 7 - Execução do comando "_stat", para apresentar as informações do ficheiro "test.txt".

Todo o código de implementação deste comando foi desenvolvido na linguagem de programação C, e encontra-se nos anexos deste documento com a designação "_stat.c".

1.7 - Comando "lista <diretoria>"

Este comando foi implementado com a designação "_ls" e, assim que executado, este verifica se há um caminho do sistema de ficheiros nos parâmetros e se este é um caminho acessível ou não, caso contrário apresenta uma mensagem de erro.

Quando este comando é executado sem qualquer caminho do sistema de ficheiros, este é executado tendo em conta o caminho atual onde executa o comando.

Este comando suporta vários caminhos de uma só execução e distingue os ficheiros quanto ao seu tipo.

Na Figura 8, pode-se observar a execução deste comando, tanto sem qualquer parâmetro, como também com um caminho válido e outro inválido.

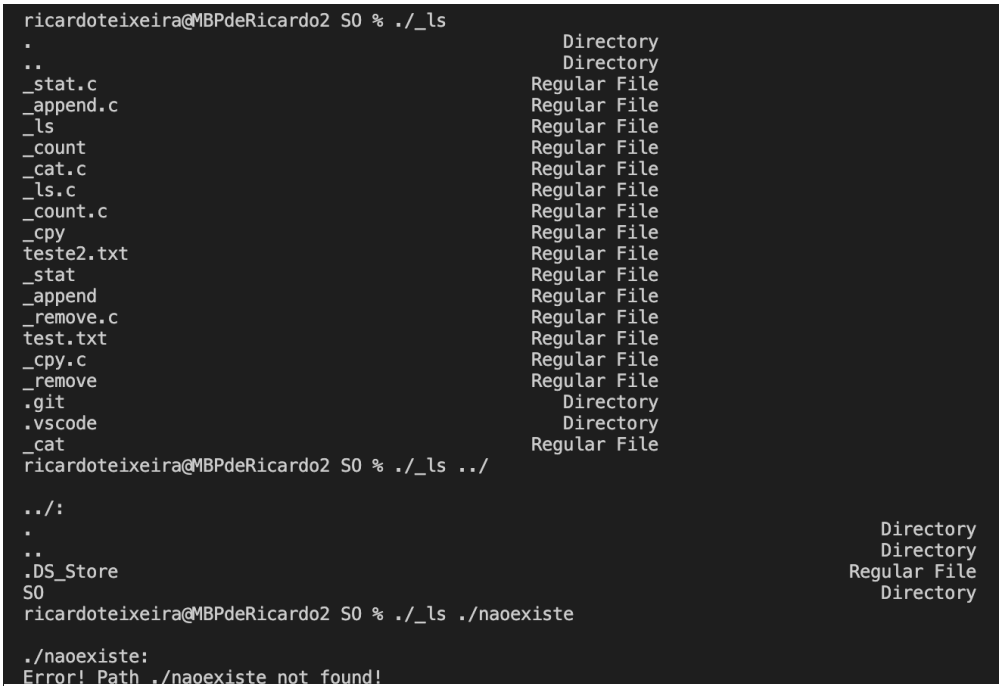

Figura 8 - Execução do comando "_ls" de variadas formas.

Todo o código de implementação deste comando foi desenvolvido na linguagem de programação C, e encontra-se nos anexos deste documento com a designação “_ls.c”.

Capítulo 2 - Manipulação de discos, volumes, partições e sistemas de ficheiros

Introdução

Neste capítulo, foi usado o sistema operativo Linux Fedora e são abordados alguns dos possíveis processos de manipulação, criação e gestão de discos, quer estes sejam físicos ou virtuais, bem como de volumes, partições e sistemas de ficheiros.

Foi proposto num servidor ou numa máquina virtual, adicionar um disco virtual com o tamanho de 10 GB (alocado dinamicamente) e posteriormente criar uma partição.

No disco virtual criado, foi proposto criar um volume que ocupe o espaço total alocado, e dentro desse mesmo volume, adicionar dois volumes lógicos, cada um com o tamanho de 5 GB.

Nesses dois volumes lógicos criados, foi ainda exigido que num deles fosse criado um sistema de ficheiros *ext4*[1] e *ext3*[2] no outro.

De seguida, foi proposto montar cada um dos sistemas de ficheiros criados nas diretorias */mnt/ext4* e */mnt/ext3*, respetivamente, ficando persistente a *reboots*.

Dentro da diretoria */mnt/ext4*, exigiu-se que fosse criado um ficheiro, em que este deve ter apenas permissões de escrita e leitura para o dono (utilizador que criou o ficheiro), o grupo não deve ter qualquer permissão e todos os outros devem ter apenas permissão de leitura.

Por fim, foi questionado acerca das permissões efetivas que o ficheiro */etc/shadow* tem e quais os utilizadores que podem escrever, ler ou executar.

As alíneas a) e d) foram resolvidas pela Bruna – 21139.
As alíneas c) e f) foram resolvidas pelo Ricardo – 20080.
As alíneas b) e e) foram resolvidas pelo Francisco – 21156.

---

[1] O *ext4* é o quarto sistema de ficheiros estendido do *Linux*. Pode-se considerar uma evolução quanto ao *ext3* em vários aspetos, como por exemplo, suporte para um sistema de ficheiros maior, melhorias na resistência à fragmentação, melhor performance, entre outros.

[2] O *ext3* é terceiro sistema de ficheiros estendido do *Linux*.

1.1 - Adicionar disco virtual e criar partição

Primeiramente, inicializou-se o processo de configuração da máquina virtual, adicionando um novo disco virtual.

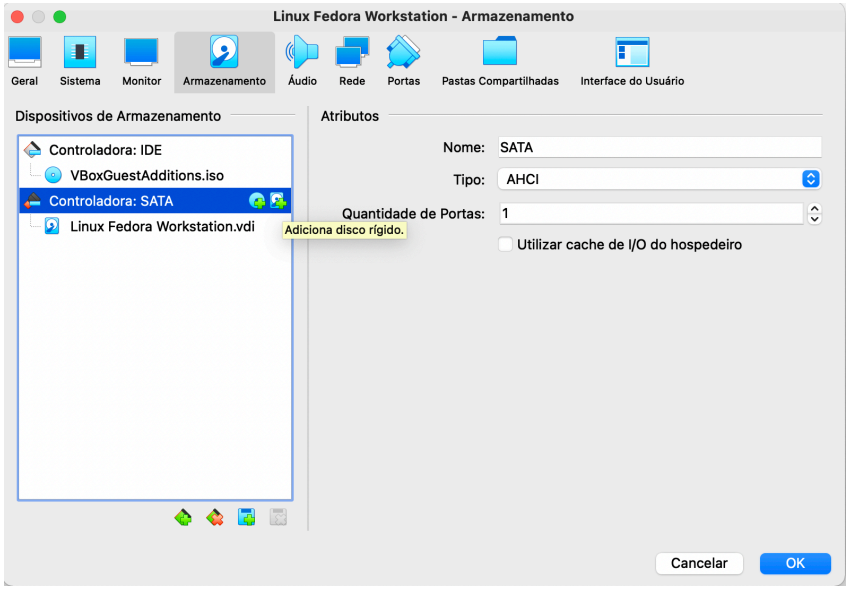

Figura 9 - Criação de disco virtual na *VirtualBox*.

Depois, selecionou-se VDI (*VirtualBox Disk Image*) como tipo de disco rígido virtual.

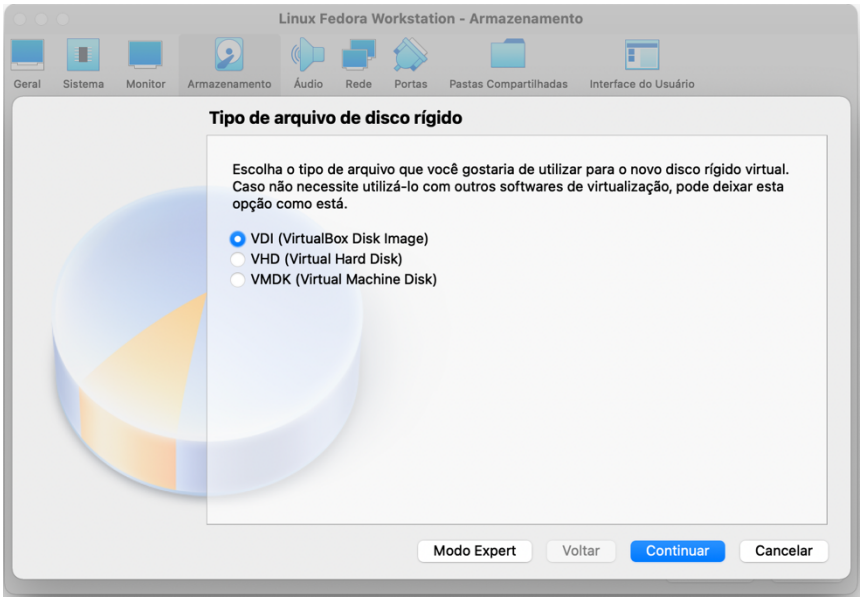

Figura 10 - Selecionar tipo de arquivo de disco rígido virtual na *VirtualBox*.

De seguida, selecionou-se “Dinamicamente alocado”, pois é um dos requisitos propostos.

Figura 11 - Selecionar e definir o tipo de armazenamento como dinamicamente alocado.

Depois, definiu-se 10 GB de tamanho do disco.

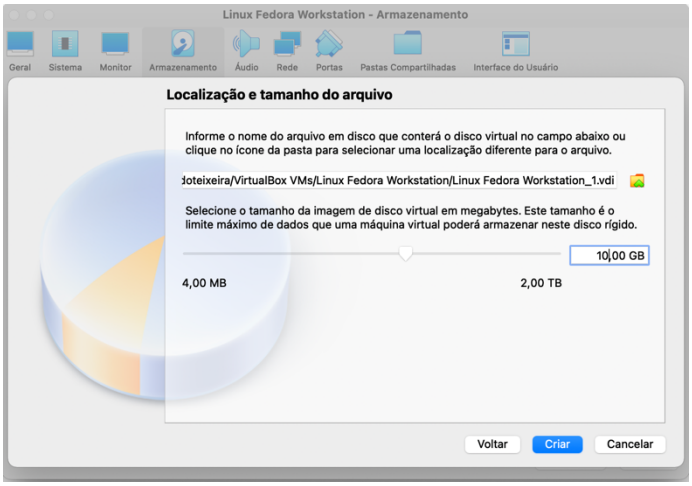

Figura 12- Definir tamanho do disco (10GB).

De seguida, anexou-se o novo disco virtual à máquina para que este seja reconhecido no sistema operativo.

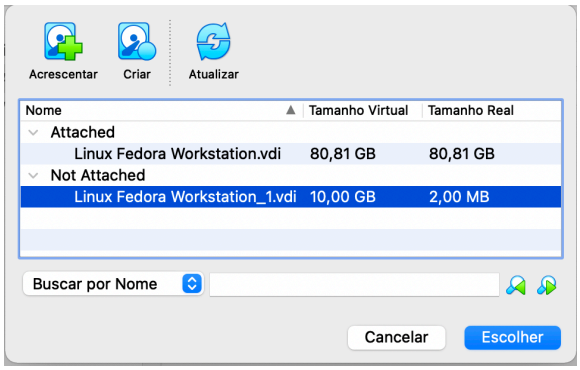

Figura 13 - Anexar disco à máquina virtual.

Já no terminal com a conta de utilizador "*root*" (conta esta que tem privilégio de administração completo), foi executado o comando:

```
sudo fdisk /dev/sdb
```

De seguida, introduziu-se a letra "n", que tem como função criar uma nova partição.

Depois, foi introduzida a letra "p", que tem como função definir essa nova partição como primária, foi também introduzido o dígito "1", simbolizando o número da partição a criar, depois pressionou-se a tecla "*enter*" duas vezes, em que uma para assumir o início (primeiro sector) da partição automaticamente, e outra para assumir o fim (último sector) da partição automaticamente, pois esta partição deve ocupar o espaço todo do disco.

Por fim, introduziu-se a letra "w", para guardar as alterações na tabela de partições do disco.

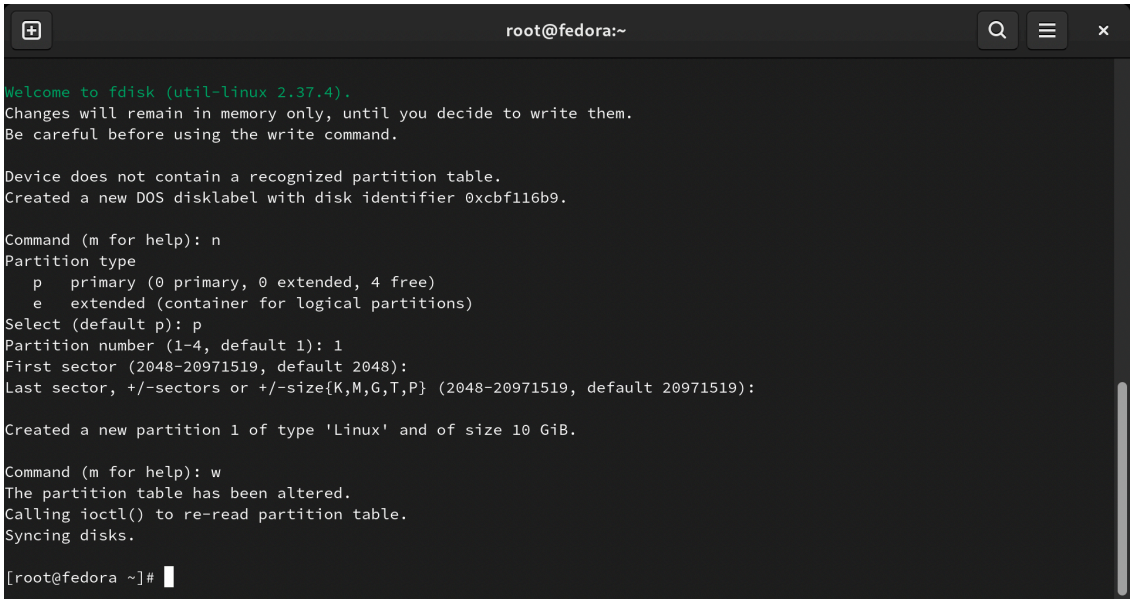

Figura 14 - Criação de partição no terminal do *Linux Fedora Workstation*.

1.2 - Criação de Volume Físico e *Volume Group*

Foi criado um Volume Físico na partição "*/dev/sdb1*" com o comando:

```
sudo pvcreate /dev/sdb1
```

Foi também criado um *Volume Group* "vgsosd" usando a partição "*/dev/sdb1*" com o comando:

```
sudo vgcreate vgsosd /dev/sdb1
```

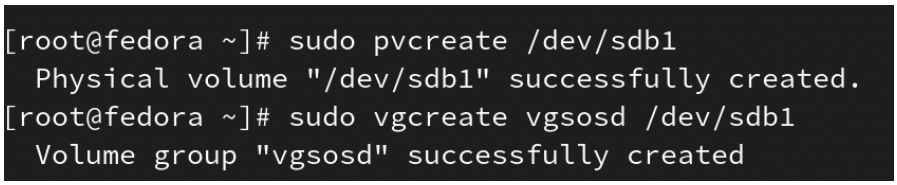

Figura 15 - Criação de Volume Físico e *Volume Group* no terminal do *Linux Fedora Workstation*.

Depois, verificou-se existência de volumes físicos com o comando:

```
sudo pvs
```

Assim como também foi verificada a existência do grupo de volumes:

```
sudo vgs
```

E pode-se verificar que ambos foram criados com sucesso.

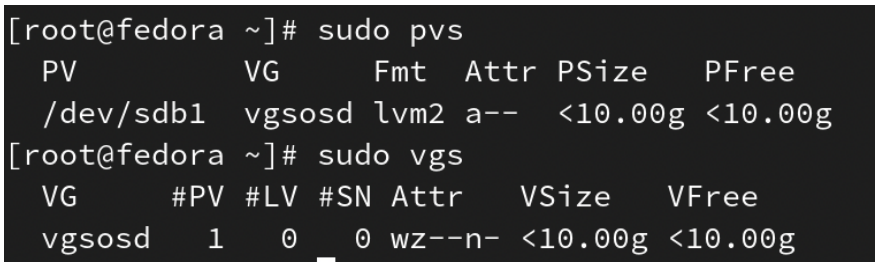

Figura 16 - Verificação de existência de Volumes Físicos e *Volume Groups*.

1.3 - Criação de Volumes Lógicos e Sistemas de Ficheiros

No volume físico anteriormente criado, foram criados dois volumes lógicos cada um com aproximadamente 5 GB de tamanho com o comando:

```
sudo lvcreate -L 5G -n lvsosd vgsosd
```

Foi criado um sistema de ficheiros "ext4", neste novo volume lógico com o seguinte comando:

```
sudo mkfs.ext4 /dev/vgsosd/lvsosd
```

```
[root@fedora ~]# sudo lvcreate -L 5G -n lvsosd vgsosd
  Logical volume "lvsosd" created.
[root@fedora ~]# sudo mkfs.ext4 /dev/vgsosd/lvsosd
mke2fs 1.46.3 (27-Jul-2021)
Creating filesystem with 1310720 4k blocks and 327680 inodes
Filesystem UUID: 787b1dd8-8e3a-414b-b155-71b76af614b5
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

Figura 17 - Criação de Volume Lógico e respetivo Sistema de Ficheiros.

O mesmo foi realizado para a outra metade do volume, criando um volume lógico usando os mesmos comandos, mas com o nome "lvsosd2".

Por lapso, foi criado com um sistema de ficheiros "ext4", quando o proposto é "ext3", então foi efetuada a correção como se pode verificar na figura a seguir, com o comando:

```
sudo mkfs.ext3 /dev/vgsosd/lvsosd2
```

```
[root@fedora ~]# sudo mkfs.ext3 /dev/vgsosd/lvsosd2
mke2fs 1.46.3 (27-Jul-2021)
/dev/vgsosd/lvsosd2 contains a ext4 file system
        created on Wed Apr  6 22:19:33 2022
Proceed anyway? (y,N) y
Creating filesystem with 1308672 4k blocks and 327680 inodes
Filesystem UUID: f24754ef-d177-4748-98ea-647954a8058f
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

Figura 18 - Configuração de novo Sistema de Ficheiros.

1.4 - Montar volumes lógicos

Montaram-se os sistemas de ficheiros acabados de criar com os comandos apresentados na figura seguinte, em que começamos por criar dois diretórios para cada volume dentro do diretório "*/mnt*", e de seguida efetuou-se realmente o "*mount*" para cada um dos volumes lógicos.

```
[root@fedora ~]# sudo mkdir /mnt/lvsosd
[root@fedora ~]# sudo mkdir /mnt/lvsosd2
[root@fedora ~]# sudo mount /dev/vgsosd/lvsosd /mnt/lvsosd
[root@fedora ~]# sudo mount /dev/vgsosd/lvsosd2 /mnt/lvsosd2
```

Figura 19 - Montagem de Volumes Lógicos.

Depois, validou-se o acesso ao novo volume, com o comando:

```
sudo mount
```

E como se pode verificar na figura seguinte, os dois volumes lógicos já aparecem na lista do "*mount*".

```
/dev/mapper/vgsosd-lvsosd on /mnt/lvsosd type ext4 (rw,relatime,seclabel)
/dev/mapper/vgsosd-lvsosd2 on /mnt/lvsosd2 type ext3 (rw,relatime,seclabel)
```

Figura 20 - Verificação dos Volumes Lógicos criados com o "*mount*".

1.5 - Permissões de ficheiros

Tal como proposto, foi criado um ficheiro com a extensão ".txt" na diretoria do volume lógico (com o sistema de ficheiros "ext4"), em que o seu nome é representado pelos números dos alunos autores deste trabalho prático.
O comando `cd /mnt/lvsosd`, permite abrir a diretoria "lvsosd", e o comando "`touch`", permite criar um ficheiro no diretório atual.

```
[root@fedora lvsosd]# cd /mnt/lvsosd
[root@fedora lvsosd]# touch 20080-21139-21156.txt
```

Figura 21- Criação de ficheiro no Volume Lógico com o Sistema de Ficheiros ".ext4".

Este ficheiro deve ter, apenas, permissões de escrita e leitura para o dono (utilizador que criou o ficheiro), o grupo não deve ter qualquer permissão neste ficheiro, e todos os outros devem ter permissão de leitura. Sendo assim, retiramos todas as permissões com o comando:

```
chmod a-r-w-x /mnt/lvsosd/20080-21139-21156.txt
```

E de seguida, executou-se os seguintes comandos:

```
// Permissão de escrita para o dono do ficheiro
     chmod u+w /mnt/lvsosd/20080-21139-21156.txt
//Permissão de leitura para o dono do ficheiro
     chmod u+r /mnt/lvsosd/20080-21139-21156.txt
//Permissão de leitura para os outros utilizadores
     chmod o+r /mnt/lvsosd/20080-21139-21156.txt
```

```
[root@fedora lvsosd]# chmod u+w /mnt/lvsosd/20080-21139-21156.txt
[root@fedora lvsosd]# chmod u+r /mnt/lvsosd/20080-21139-21156.txt
[root@fedora lvsosd]# ls -la
total 20
drwxr-xr-x. 3 root root  4096 Apr  6 22:24 .
drwxr-xr-x. 1 root root    26 Apr  6 22:22 ..
-rw-------. 1 root root     0 Apr  6 22:24 20080-21139-21156.txt
drwx------. 2 root root 16384 Apr  6 22:17 lost+found
[root@fedora lvsosd]# chmod o+r /mnt/lvsosd/20080-21139-21156.txt
[root@fedora lvsosd]# ls -la
total 20
drwxr-xr-x. 3 root root  4096 Apr  6 22:24 .
drwxr-xr-x. 1 root root    26 Apr  6 22:22 ..
-rw----r--. 1 root root     0 Apr  6 22:24 20080-21139-21156.txt
drwx------. 2 root root 16384 Apr  6 22:17 lost+found
```

Figura 22 - Definição de permissões no ficheiro.

Conclusão

Este trabalho prático foi importante para consolidar os conteúdos lecionados na Unidade Curricular de Sistemas Operativos, pois é na parte prática que os reais desafios aparecem. Além disso fortalece a entreajuda entre colegas e aprendem-se outros conteúdos extra-aulas.

O trabalho desenvolvido, também foi importante para perceber como os comandos dos Sistemas Operativos funcionam, principalmente os comandos do sistema operativo *Linux*, ou até de outros sistemas operativos Unix como *macOS* e mesmo do sistema operativo *Windows.*

Pode-se concluir ainda, que consolidou-se mais conhecimento acerca dos sistemas de ficheiros e gestão de discos no sistema operativo.

Bibliografia

- Stat(2) – Linux Manual Page - https://man7.org/linux/man-pages/man2/lstat.2.html - Acedido a 27/04/2022
- Opendir(3) – Linux Manual Page - https://man7.org/linux/man-pages/man3/opendir.3.html - Acedido a 27/04/2022
- Readdir(3) – Linux Manual Page - https://man7.org/linux/man-pages/man3/readdir.3.html - Acedido a 27/04/2022
- Unlink(2) – Linux Manual Page - https://man7.org/linux/man-pages/man2/unlink.2.html - Acedido a 27/04/2022
- Getpwuid(3) – Password File Entry (Linux Manual Page) - https://linux.die.net/man/3/getpwuid - Acedido a 27/04/2022

Anexos

_append.c

```c
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

int main (int argc, char *argv[])
{
    int source, dest;
    char buffer;
    char filename[200];

    if ((argc < 3) && ((argc-1) % 2 != 0))
    {
        write(STDERR_FILENO, "Error! You should give at least 2 filenames or
an even number of filenames!\n", 76);
        return 0;
    }

    for (int i = 1; i < argc; i = i + 2)
    {
        source = open(argv[i], O_RDONLY);
        dest = open(argv[i + 1], O_WRONLY | O_APPEND);
        if (source == -1)
        {
            write(STDERR_FILENO, "Error! File ", 13);
            write(STDERR_FILENO, argv[i], strlen(argv[i]));
            write(STDERR_FILENO, " not found!\n", 13);
            close(source);
            continue;
        }
        if (dest == -1)
        {
            write(STDERR_FILENO, "Error! File ", 13);
            write(STDERR_FILENO, argv[i + 1], strlen(argv[i + 1]));
```

```
            write(STDERR_FILENO, " not found!\n", 13);
            close(dest);
            continue;
        }

        while(read(source, &buffer, 1))
        {
            write(dest, &buffer, 1);
        }

        write(STDOUT_FILENO, "File ", 5);
        write(STDOUT_FILENO, argv[i + 1], strlen(argv[i + 1]));
        write(STDOUT_FILENO, " modified successfully!\n", 24);
        close(source);
        close(dest);
    }

    return 0;
}
```

_cat.c

```c
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int fd, len;
    char buffer;


    if (argc < 2)
    {
        write(STDERR_FILENO, "Error! No files given, please insert a
filename!\n", 49);
        return 0;
    }

    for (int i = 1; i < argc; i++)
    {
        fd = open(argv[i], O_RDONLY);
        if (fd == -1)
        {
            write(STDERR_FILENO, "Error! File ", 13);
            write(STDERR_FILENO, argv[i], strlen(argv[i]));
            write(STDERR_FILENO, " not found!\n", 13);
        }
        else
        {
            write(STDOUT_FILENO, argv[i], strlen(argv[i]));
            write(STDOUT_FILENO, ":\n", 2);
            while(read(fd, &buffer, 1))
            {
                write(STDOUT_FILENO, &buffer, 1);
            }
            write(STDOUT_FILENO, "\n", 1);
        }
    }   close(fd);
    return 0;
}
```

_count.c

```c
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int fd;
    int counter = 0;
    char buffer;


    if (argc < 2)
    {
        write(STDERR_FILENO, "Error! No files given, please insert a
filename!\n", 49);
        return 0;
    }

    for (int i = 1; i < argc; i++)
    {
        fd = open(argv[i], O_RDONLY);
        if (fd == -1)
        {
            printf("Error! File %s not found!\n", argv[i]);
            // write(STDERR_FILENO, "Error! File ", 13);
            // write(STDERR_FILENO, argv[i], strlen(argv[i]));
            // write(STDERR_FILENO, " not found!\n", 13);
        }
        else
        {
            while(read(fd, &buffer, 1))
            {
                if (buffer == '\n')
                    counter++;
            }

            printf("%d line(s) counted on file: %s\n", counter, argv[i]);

            counter = 0;
        }
    }   close(fd);
    return 0;
}
```

_cpy.c

```c
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
    int source, dest;
    char buffer;
    char filename[200];

    if (argc < 2 )
    {
        write(STDERR_FILENO, "Error! No files given, please insert filename!
\n", 49);
        return 0;
    }
    for (int i = 1; i < argc; i++)
    {
        source = open(argv[i], O_RDONLY);

        if (source == -1)
        {
            write(STDERR_FILENO, "Error! File ", 13);
            write(STDERR_FILENO, argv[i], strlen(argv[i]));
            write(STDERR_FILENO, " not found!\n", 13);
        }
        else
        {
            strcpy(filename, argv[i]);
            strcat(filename, ".copia");
            dest = open(filename, O_CREAT | O_TRUNC | O_WRONLY | S_IRUSR |
S_IWUSR | S_IRGRP | S_IROTH);
            while(read(source, &buffer, 1))
            {
                write(dest, &buffer, 1);
            }
            write(STDOUT_FILENO, "File ", 5);
            write(STDOUT_FILENO, argv[i], strlen(argv[i]));
            write(STDOUT_FILENO, " copied successfully!\n", 22);
            close(source);
            close(dest);
        }
    }
    return 0;
}
```

_ls.c

```c
#include <fcntl.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/types.h>
#include <string.h>
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int fd;
    DIR *dir;
    struct dirent *dent;
    char ftype[50];

    if (argc < 2)
    {
        dir = opendir("./");
        while ((dent = readdir(dir)) != NULL)
        {
            switch (dent->d_type)
            {
            case DT_BLK:
                strcpy(ftype, "Block Device");
                break;
            case DT_CHR:
                strcpy(ftype, "Character Device");
                break;
            case DT_DIR:
                strcpy(ftype, "Directory");
                break;
            case DT_FIFO:
                strcpy(ftype, "FIFO/pipe");
                break;
            case DT_LNK:
                strcpy(ftype, "Symlink");
                break;
            case DT_REG:
                strcpy(ftype, "Regular File");
                break;
            case DT_SOCK:
                strcpy(ftype, "Socket");
                break;
            default:
                strcpy(ftype, "Unknown Type");
                break;
            }
            printf("%-30s%30s\n", dent->d_name, ftype);
```

```
        }   closedir(dir);
    }

    for (int i = 1; i < argc; i++)
    {
        dir = opendir(argv[i]);
        printf("\n%s:\n", argv[i]);
        if (dir == NULL)
        {
            printf("Error! Path %s not found!\n", argv[i]);
            break; // Break to not execute closedir(dir); because there's not
any open dir and it could cause segmentation fault
        }
        else
        {
            while ((dent = readdir(dir)) != NULL)
            {
                switch (dent->d_type)
                {
                case DT_BLK:
                    strcpy(ftype, "Block Device");
                    break;
                case DT_CHR:
                    strcpy(ftype, "Character Device");
                    break;
                case DT_DIR:
                    strcpy(ftype, "Directory");
                    break;
                case DT_FIFO:
                    strcpy(ftype, "FIFO/pipe");
                    break;
                case DT_LNK:
                    strcpy(ftype, "Symlink");
                    break;
                case DT_REG:
                    strcpy(ftype, "Regular File");
                    break;
                case DT_SOCK:
                    strcpy(ftype, "Socket");
                    break;
                default:
                    strcpy(ftype, "Unknown Type");
                    break;
                }
                printf("%-70s%20s\n", dent->d_name, ftype);
            }
        }   closedir(dir);
    }
    return 0;
}
```

_remove.c

```c
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

int main(int argc, char const *argv[])
{
    int fd, len;
    char content[300];


    if (argc < 2)
    {
        write(STDERR_FILENO, "Error! No files given, please insert a
filename!\n", 49);
        return 0;
    }

    for (int i = 1; i < argc; i++)
    {
        fd = access(argv[i], F_OK);
        if (fd == -1)
        {
            write(STDERR_FILENO, "Error! File ", 13);
            write(STDERR_FILENO, argv[i], strlen(argv[i]));
            write(STDERR_FILENO, " not found!\n", 13);
        }
        else
        {
            unlink(argv[i]);
            write(STDOUT_FILENO, content, len);
            write(STDOUT_FILENO, "File ", 5);
            write(STDOUT_FILENO, argv[i], strlen(argv[i]));
            write(STDOUT_FILENO, " removed successfully!\n", 23);
        }
    }   close(fd);

    return 0;
}
```

_stat.c

```c
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <time.h>
#include <sys/stat.h>
#include <pwd.h>
#include <stdio.h>

/**
 * @brief Get the Username from uid
 *
 * @param uid
 * @return const char*
 */
const char *getUsername(uid_t uid)
{
    struct passwd *pw = getpwuid(uid);
    if (pw)
    {
        return pw->pw_name;
    }
    return "";
}

int main(int argc, char const *argv[])
{
    int fd;
    struct stat stats;

    if (argc < 2)
    {
        write(STDERR_FILENO, "Error! No paths given, please insert a
pathname!\n", 49);
        return 0;
    }

    for (int i = 1; i < argc; i++)
    {
        fd = stat(argv[i], &stats);
        printf("\n%s:\n", argv[i]);
        if (fd == -1)
        {
            write(STDERR_FILENO, "Error! Path ", 13);
            write(STDERR_FILENO, argv[i], strlen(argv[i]));
            write(STDERR_FILENO, " not found!\n", 13);
        }
        else
```

```
        {
            printf("File type:\t\t");
            switch (stats.st_mode & S_IFMT)
            {
                case S_IFBLK:
                    printf("block device\n");
                    break;
                case S_IFCHR:
                    printf("character device\n");
                    break;
                case S_IFDIR:
                    printf("directory\n");
                    break;
                case S_IFIFO:
                    printf("FIFO/pipe\n");
                    break;
                case S_IFLNK:
                    printf("symlink\n");
                    break;
                case S_IFREG:
                    printf("regular file\n");
                    break;
                case S_IFSOCK:
                    printf("socket\n");
                    break;
                default:
                    printf("unknown?\n");
                    break;
            }

            printf("I-node number:\t\t%ld\n", (long) stats.st_ino);
            printf("Ownership: %s\t\tUID: %ld   GID: %ld\n",
getUsername(stats.st_uid), (long) stats.st_uid, (long) stats.st_gid);
            printf("Last status change:\t\t%s", ctime(&stats.st_ctime));
            printf("Last file access:\t\t%s", ctime(&stats.st_atime));
            printf("Last file modification:\t\t%s", ctime(&stats.st_mtime));

        }   close(fd);
    }

    return 0;
}
```