**Instituto Politécnico do Cavado e do Ave Curso Engenharia de Sistemas Informáticos Sistemas Operativos**


**Autores:**

Nuno Santos N 21132

Pedro Pereira N 21114

Pedro Nascimento N 21108

André Freitas N 21112

Luís Silva N 21113

# Índice



# Índice de figuras

Distribuição do trabalho

- André Freitas: Parte 2 – Alíneas b), c), d), e)
- Luís Silva: Parte 1 – Alíneas e), f), g)
- Nuno Santos: Parte 1 – Alíneas a), b)
- Pedro Nascimento: Parte 1 – Alínea d);
  Parte 2 – Alínea f)
- Pedro Pereira: Parte 1 – Alínea c);
  Parte 2 – Alínea a)

Parte 1

# Implementação de um conjunto de comandos para manipulação de ficheiros

Era pretendido que se implementasse os seguintes comandos:

a) **Mostra ficheiro** – Este comando deve apresentar no ecrã (todo) o conteúdo do ficheiro indicado como parâmetro. Caso o ficheiro não exista, o comando deve avisar o utilizador que o ficheiro não existe;
b) **Copia ficheiro** – Este comando deve criar um novo ficheiro, cujo nome é "ficheiro.copia", cujo conteúdo é uma cópia de (todo) o conteúdo do ficheiro passado como parâmetro no comando, com o nome ficheiro. Caso o ficheiro não exista, deve ser apresentado um aviso ao utilizador;
c) **Acrescenta origem destino** – Este comando deve acrescentar (todo) o conteúdo da "origem" no final do "destino". Caso algum dos ficheiros não exista, deve ser apresentado um aviso ao utilizador;
d) **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;
e) **Apaga ficheiro** – Este comando deve apagar o ficheiro com o nome indicado. No caso de o ficheiro indicado não existir, e apenas, deve ser apresentado um aviso ao utilizador;
f) **Informa ficheiro** – Este comando apresenta apenas a informação do sistema de ficheiros em relação ao ficheiro indicado, tipo de ficheiro (normal, diretoria, link, etc.), i-node, utilizador dono em formato textual e datas de criação, leitura e modificação em formato textual;
g) **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. Adicionalmente, deve distinguir ficheiros simples de diretorias através de uma indicação textual.

Através da chamada de funções ao sistema (system calls). Estes comandos implementados em C serão invocados através de um interpretador de comandos.

```c
void Mostra(char *nome){
    int ficheiro, tamanho;
    char linhas[999];
    ficheiro = open(nome, O_RDONLY);

    if (ficheiro == -1){
        perror("\nFicheiro nao encontrado");
        exit(1);
    }

    tamanho = read (ficheiro, linhas, sizeof(linhas));
    write(STDOUT_FILENO, linhas, tamanho);
    close(ficheiro);
}
```

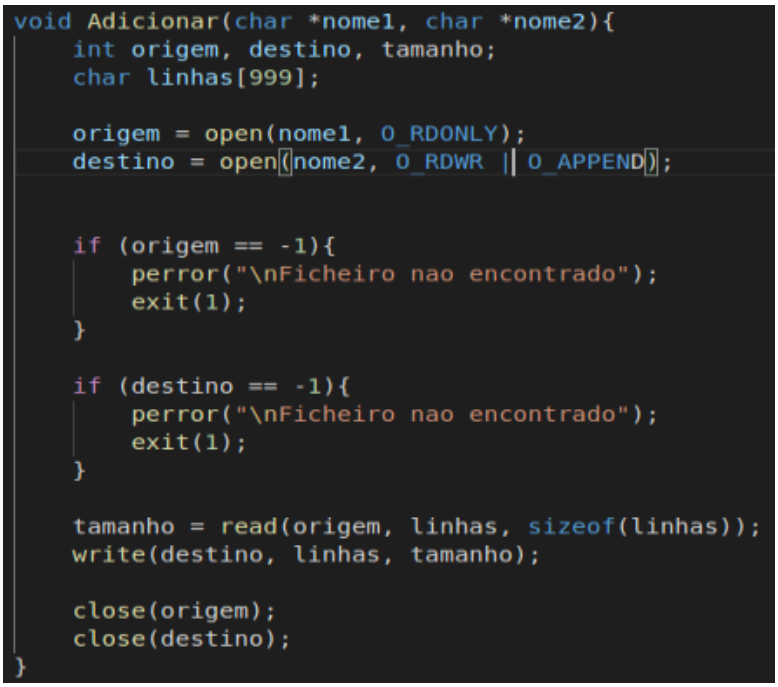
*Figura 1: Mostra ficheiro*

O programa recebe como argumento um ficheiro verificando sempre se realmente recebeu corretamente os argumentos requisitados. De seguida executa a função e apresenta todo o conteúdo do ficheiro indicado como parâmetro.

```c
void Copiar(char *nome){
    int ficheiro, fd, tamanho;

    mode_t modo =  S_IRUSR | S_IWUSR | S_IXUSR;
    char *nome_ficheiro = "/home/andre/Desktop/SO/ficheiro.copia";
    creat(nome_ficheiro, modo);

    char linhas[999];

    ficheiro = open(nome, O_RDONLY);
    read(ficheiro, linhas, tamanho);

    if (ficheiro == -1){
        perror("Ficheiro nao encontrado");
        exit(1);
    }

    fd = open("ficheiro.copia", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);

    tamanho = read(ficheiro, linhas, sizeof(linhas));
    write(fd, linhas, tamanho);

    close(fd);
    close(ficheiro);
}
```

*Figura 2: Copiar ficheiro*

O programa recebe como argumento o ficheiro em questão verificando sempre se realmente recebeu corretamente os argumentos requisitados. Este comando cria um novo ficheiro, cujo conteúdo é uma cópia de todo o conteúdo do ficheiro passado como parâmetro no comando.



```
void Adicionar(char *nome1, char *nome2){
    int origem, destino, tamanho;
    char linhas[999];

    origem = open(nome1, O_RDONLY);
    destino = open(nome2, O_RDWR | O_APPEND);


    if (origem == -1){
        perror("\nFicheiro nao encontrado");
        exit(1);
    }

    if (destino == -1){
        perror("\nFicheiro nao encontrado");
        exit(1);
    }

    tamanho = read(origem, linhas, sizeof(linhas));
    write(destino, linhas, tamanho);

    close(origem);
    close(destino);
}
```

*Figura 3: Adicionar*

Este comando recebe um ficheiro de origem e um ficheiro de destino. Além disso, faz tambem a verificação da existencia dos dois ficheiros anteriormente mencionados. Por fim, determina o tamanho do ficheiro de origem e o seu número de linhas e escreve no ficheiro de destino o que foi determinado no ficheiro de origem. No final da função fecha ambos os ficheiros.

```c
void Contar(char *nome){
    char linhas[999];
    int tamanho, ficheiro;
    int contador = 0;

    ficheiro = open(nome, O_RDONLY);

    if (ficheiro == -1){
        perror("\nFicheiro nao encontrado");
        exit(1);
    }

    tamanho = read(ficheiro, linhas, sizeof(linhas));

    for (int i=0; i<tamanho; i++){
        if (linhas[i] == '\n'){
            contador++;
        }
    }

    printf("\n\nO ficheiro %s tem %d linhas\n", nome, contador);
    close(ficheiro);
}
```

*Figura 4: Contar*

Este comando recebe um ficheiro fazendo a verificação da existência do mesmo. Após verificação determina o tamanho do ficheiro e o seu número de linhas fazendo a contagem dessas linhas do ficheiro.

```c
void Apagar(char *nome){
    int ficheiro;

    ficheiro = open(nome, O_RDONLY);

    if (ficheiro == -1){
        perror("\nFicheiro nao encontrado");
        exit(1);
    }

    close(ficheiro);
    int apagar;
    apagar = unlink(nome);
}
```

*Figura 5: Apagar*

Esta função recebe um ficheiro como argumento, verifica a existência desse mesmo ficheiro. Após essa verificação o ficheiro em questão é apagado.

```c
void Informar(char *nome){
    struct stat informacao;
    struct passwd *pwd;
    int ficheiro;

    ficheiro = open(nome, O_RDONLY);
    stat(nome, &informacao);

    printf("Nome do ficheiro: %s\n", nome);
    printf("Inode: %lu\n", informacao.st_ino);
    printf("Tipo: ");
    if(S_ISREG(informacao.st_mode)){
        printf("Ficheiro");
    }
    if(S_ISDIR(informacao.st_mode)){
        printf("Diretoria");
    }
    if(S_ISLNK(informacao.st_mode)){
        printf("Link");
    }
    if(S_ISFIFO(informacao.st_mode)){
        printf("Pipe");
    }
    printf("\n");

    pwd = getpwuid(informacao.st_uid);
    if (pwd == NULL) {
      perror("getpwuid");
      exit(EXIT_FAILURE);
    }
    printf("Nome utilizador: %s\n", pwd->pw_name);
    printf("Data de criação: %s", ctime(&informacao.st_ctime));
    printf("Data de modificação: %s", ctime(&informacao.st_mtime));
    printf("Data de leitura: %s", ctime(&informacao.st_atime));

    close(ficheiro);
}
```

*Figura 6: Informar*

Este comando começa por abrir um ficheiro pré-selecionado e através do uso da função “stat ()” passa a listar informação relativa ao ficheiro em questão.

```c
void Listar(){
    DIR *d;

    struct dirent *dir;

    d = opendir(".");

    if (d){
        while((dir = readdir(d)) != NULL){
            printf("\n%s\n", dir->d_name);
        }

        closedir(d);
    }
}
```

*Figura 7: Listar*

Este comando abre a diretoria onde o ficheiro pedido se encontra e apresenta uma lista de todas as pastas e ficheiros existentes nessa diretoria.

Parte 2

## Implementação de um conjunto de comandos para a manipulação de ficheiros

Seguindo uma série de instruções foi possível aplicar todos os conhecimentos adquiridos durante o semestre sobre a manipulação de ficheiros.

**a) Num servidor virtual, adicione um disco novo com o tamanho de 10GB (espaço alocado dinamicamente) e crie uma partição.**

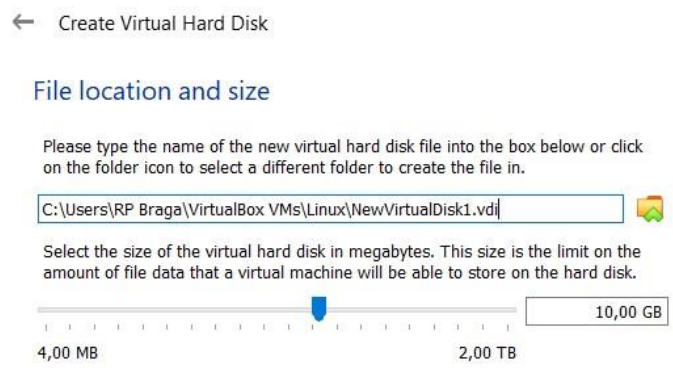




*Figura 8: Criação do disco (Parte2)*

*Figura 9: Criação do disco (Parte 1)*

```
Comando (m para ajuda): n
Tipo da partição
   p   primária (0 primárias, 0 estendidas, 4 livre)
   e   estendida (recipiente para partições lógicas)
Selecione (padrão p): p
Número da partição (1-4, padrão 1): 1
Primeiro setor (2048-20971519, padrão 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-20971519, padrão 20971519):

Criada uma nova partição 1 do tipo "Linux" e de tamanho 10 GiB.

Comando (m para ajuda): w
A tabela de partição foi alterada.
Chamando ioctl() para reler tabela de partição.
Sincronizando discos.
```

*Figura 10: Criação de uma partição*

Para a criação do disco foi necessário aceder às definições da virtual box e criar manualmente um disco com 10gb, como pedido.
Depois através da linha de comandos criou-se uma partição primária que ocupa todo disco. Para isso foram usados os comandos:

- sudo fdisk /dev/sdb n
- p
- 1
- Enter x2
- w

**b) No disco virtual criado na alínea a), deve criar um volume, que ocupe o espaço todo, e dentro desse volume, deve adicionar dois volumes lógicos, cada um com o tamanho de 5GB.**

```
fura@Fura-VirtualBox:~$ sudo pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created.
fura@Fura-VirtualBox:~$ sudo vgcreate vgsosd /dev/sdb1
  Volume group "vgsosd" successfully created
```

*Figura 11: Criação de um volume*

```
fura@Fura-VirtualBox:~$ sudo lvcreate -L 5G -n lv1 vgsosd
  Logical volume "lv1" created.
fura@Fura-VirtualBox:~$ sudo lvcreate -L 5G -n lv2 vgsosd
  Volume group "vgsosd" has insufficient free space (1279 extents): 1280 required.
fura@Fura-VirtualBox:~$ sudo lvcreate -L 4.99G -n lv2 vgsosd
  Rounding up size to full physical extent 4,99 GiB
  Logical volume "lv2" created.
```

*Figura 12: Criação de dois volumes lógicos*

Pela consola criamos um volume que ocupa a partição criada anteriormente. De seguida foram criados dois de volumes lógicos de 5gb cada um. Para a realização desta tarefa foi necessário estes comandos:

- sudo pvcreate /dev/sdb1
- sudo vgcreate vgsosd /dev/sdb1
- sudo lvcreate -L 5G -n lv1 vgsosd
- sudo lvcreate -L 5G -n lv2 vgsosd

**c) Nos volumes lógicos criados no passo b), crie um sistema de ficheiros ext4 em um deles e ext3 no outro.**

**d) Monte cada um dos sistemas de ficheiros criados em c) nas directorias /mnt/ext4 e /mnt/ext3, respectivamente, ficando persistente a reboots.**

```
fura@Fura-VirtualBox:~$ sudo mkfs.ext4 /dev/vgsosd/lv1
mke2fs 1.45.5 (07-Jan-2020)
A criar sistema de ficheiros com 1310720 4k blocos e 327680 inodes
UUID do sistema de ficheiros: 75487174-589d-48a6-be35-e04410f76e6a
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

fura@Fura-VirtualBox:~$ sudo mkdir /mnt/lv1
```

*Figura 17: Criação do sistema de ficheiros ext4*

```
fura@Fura-VirtualBox:/mnt$ sudo mount /dev/vgsosd/lv1 /mnt/lv1
fura@Fura-VirtualBox:/mnt$ sudo mount
```

*Figura 15: Montagem do sistema de ficheiros ext4*

```
fura@Fura-VirtualBox:~$ sudo mkfs.ext3 /dev/vgsosd/lv2
mke2fs 1.45.5 (07-Jan-2020)
A criar sistema de ficheiros com 1308672 4k blocos e 327680 inodes
UUID do sistema de ficheiros: e8a114a8-ddac-485f-bfe3-fc6db600bea0
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

fura@Fura-VirtualBox:~$ sudo mkdir /mnt/lv2
fura@Fura-VirtualBox:~$ sudo mount -t ext3 /dev/vgsosd/lv2 /mnt/lv2
fura@Fura-VirtualBox:~$ sudo mount
```

*Figura 16: Criação e montagem do sistema de ficheiros ext3*

Para a criação e a montagem destes sistemas de ficheiros usamos:

- sudo mkfs.ext4(3) /dev/vgsosd/lv1(2)
- sudo mkdir /mnt/lv1(2)
- sudo mount /dev/vgsosd/lv1(2) /mnt/lv1(2)

e) **Dentro da diretoria /mnt/ext4, crie um ficheiro com o nome composto pelo grupo dos números de alunos que constituem o trabalho, e a extensão .txt (exemplo: 22222-22233-23333-24003.txt).**
**Esse ficheiro deverá ter, apenas, permissões de escrita e leitura para o dono (que será o utilizador que está a usar o sistema sem ser root), o grupo não deve ter qualquer permissão neste ficheiro, e todos os outros devem ter permissão de leitura.**


*Figura 17: Criação de um ficheiro de texto*

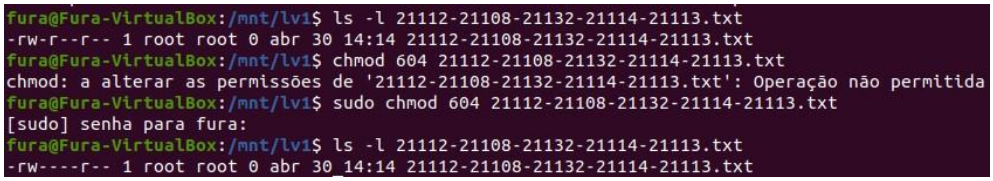



*Figura 18: Alteração das permissões*

Foi necessário criar um ficheiro de texto cujo nome fosse os números dos alunos do grupo. Para isso foi usado a função “touch ()”:

- sudo touch 21112-21108-21132-21114-21113.txt

Depois era pedido a mudança das permissões conforme o enunciado. Para isso foi usado um sistema de números que representam as permissões.

| Octal Value | Read | Write | Execute |
|---|---|---|---|
| 7 | r | w | x |
| 6 | r | w | - |
| 5 | r | - | x |
| 4 | r | - | - |
| 3 | - | w | x |
| 2 | - | w | - |
| 1 | - | - | x |
| 0 | - | - | - |

- chmod 604 21112-21108-21132-21114-21113.txt

**f) Quais as permissões efetivas que o ficheiro /etc/shadow tem? Indique quais os utilizadores que podem escrever nele, ler ou executá-lo.**

```
fura@Fura-VirtualBox:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1460 abr 30 10:33 /etc/shadow
```

*Figura 19: Permissões do ficheiro /etc/shadow*

O ficheiro /etc/shadow tem permissões 640.
Isto significa que o user (normalmente o user root) consegue ler e escrever, o grupo apenas pode ler e os outros não têm qualquer tipo de permissão no ficheiro.