Relatório do trabalho da disciplina de Sistemas Operativos

Trabalho Prático

João Carvalho – 12747

Pedro Costa - 13747

Joel Faria - 28001

Diogo Graça – 28004

Nuno Silva - 28005

Licenciatura em Engenharia de Sistemas Informáticos

Maio de 2024

|  |
| --- |
| Afirmo por minha honra que não recebi qualquer apoio não autorizado na realização deste trabalho prático. Afirmo igualmente que não copiei qualquer material de livro, artigo, documento web ou de qualquer outra fonte exceto onde a origem estiver expressamente citada. |

João Carvalho – 12747

Pedro Costa - 13747

Joel Faria - 28001

Diogo Graça – 28004

Nuno Silva - 28005

Índice

[Introdução 1 1](#_Toc167455727)

[Desenvolvimento 2 3](#_Toc167455728)

[Parte 3 - A 3](#_Toc167455729)

[Parte 3 – B 9](#_Toc167455730)

[Conclusão 11](#_Toc167455731)

# Introdução 1

Este trabalho prático tem como objetivo a aplicação prática dos conceitos de gestão de processos e de ficheiros, no âmbito da disciplina de Sistemas Operativos. O trabalho prático está dividido em 3 partes com o objetivo de aplicarmos os conhecimentos adquiridos em algumas das áreas que foram abordadas ao longo do semestre nesta unidade curricular, nomeadamente o conceito de gestão de processos e de gestão de ficheiros.O Pedro Costa contribui para o desenvolvimento da Parte 1, o Joel Faria, o Diogo Graça, e o Nuno Silva para a Parte 2 e o João Carvalho para a Parte 3.

# Desenvolvimento 2

## Parte 1:

O código está estruturado em três ficheiros:

main.c - Ficheiro C que apresenta um menu de opções ao utilizador e chama as funções apropriadas conforme a opção que o utilizador selecione.

funcoes.h – Ficheiro com a definição das funções auxiliares e das funções principais utilizadas no código. Além disso, define constantes como BUFFER\_SIZE e FICHEIRO\_SIZE, que são usadas para manipular buffers e tamanhos de nomes de ficheiros

funcoes.c – Ficheiro C com a implementação de todas as funções definidas em funcoes.h. Utiliza chamadas ao sistema para realizar operações de entrada e saída de ficheiros, leitura de diretórios, e obtenção de informações sobre ficheiros.

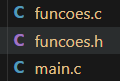


Figura 1 - Estrutura do código

Descrição das Funções Implementadas

1. mostraFicheiro

Esta função abre um ficheiro em modo de leitura e exibe o conteúdo do ficheiro no ecrã. Se o ficheiro não existir aparece uma mensagem de erro.

- Usa ‘open’ para abrir o ficheiro em modo de leitura (‘O\_RDONLY’).

- Usa ‘read’ para ler o ficheiro em blocos definidos pela variável ‘BUFFER\_SIZE’.

- Usa ‘write’ para exibir o conteúdo no ecrã.

- Inclui verificações de erro para assegurar que o ficheiro foi aberto e lido corretamente.

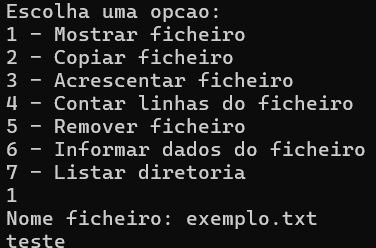


Figura 2 - Exemplo Função MostraFicheiro()

2. copiaFicheiro

Esta função cria uma cópia de um ficheiro. O nome do ficheiro de cópia é o nome do ficheiro original com a adição de ".copia". Se o ficheiro original não existir aparece uma mensagem de erro.

- Abre o ficheiro de origem em modo de leitura (‘O\_RDONLY’) e o ficheiro de destino em modo de escrita (‘O\_WRONLY | O\_CREAT | O\_TRUNC’).

- Usa ‘read’ e ‘write’ para copiar o conteúdo do ficheiro de origem para o ficheiro de destino.

- Inclui verificações de erro para assegurar que os ficheiros foram abertos e copiados corretamente.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 3 - Exemplo Função CopiaFicheiro()

3. acrescentaFicheiro

Esta função acrescenta o conteúdo de um ficheiro de origem ao final de um ficheiro de destino. Se algum dos ficheiros não existir aparece uma mensagem de erro.

- Abre o ficheiro de origem em modo de leitura (‘O\_RDONLY’) e o ficheiro de destino em modo de escrita (`O\_WRONLY | O\_CREAT | O\_APPEND`).

- Usa ‘read’ e ‘write’ para ler o conteúdo do ficheiro de origem e escrevê-lo no final do ficheiro de destino.

- Inclui verificações de erro para assegurar que os ficheiros foram abertos e escritos corretamente.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Figura 4 – Exemplo Função AcrescentaFicheiro ()

Uma imagem com texto, Tipo de letra, captura de ecrã, design

Descrição gerada automaticamente

Figura 5 - Resultado ficheiro de saida

4. contaFicheiro

Esta função conta o número de linhas em um ficheiro. Se o ficheiro não existir aparece uma mensagem de erro.

- Abre o ficheiro em modo de leitura (‘O\_RDONLY’).

- Usa ‘read’ para ler o ficheiro em blocos.

- Conta as linhas percorrendo o buffer lido e conta o número caracteres com uma nova linha (‘\n’).

- Inclui verificações de erro para assegurar que o ficheiro foi aberto e lido corretamente.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 6 - Exemplo Função ContaFicheiro()

5. removeFicheiro

Esta função apaga um ficheiro. Se o ficheiro não existir aparece uma mensagem de erro.

- Usa ‘unlink’ para remover o ficheiro especificado.

- Verifica se o ficheiro foi removido com sucesso e exibe uma mensagem de erro caso contrário.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 7 - Exemplo Função RemoveFicheiro()

6. informaFicheiro

Esta função apresenta informações sobre um ficheiro.

- Usa ‘stat’ para obter informações detalhadas sobre o ficheiro.

- Mostra informações como tipo de ficheiro, inode, proprietário, e datas de criação, leitura e modificação.

- Usa ‘strftime’ para formatar as datas de forma legível.

- Verifica se a chamada ‘stat’ foi bem-sucedida antes de exibir as informações.

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Figura 8 - Exemplo Função InformaFicheiro()

7. listaDiretoria

Esta função lista o conteúdo de uma diretoria, distinguindo ficheiros simples de diretórios.

- Usa ‘opendir’ para abrir a diretoria e ‘readdir’ para ler o conteúdo.

- Usa ‘stat’ para distinguir entre ficheiros simples e diretórios.

- Exibe os nomes dos ficheiros e dos diretórios, ignorando as entradas "." e "..".

- Inclui verificações de erro para assegurar que a diretoria foi aberta corretamente.

Uma imagem com texto, captura de ecrã, Tipo de letra, design

Descrição gerada automaticamente

Figura 9 - Exemplo Função ListaDiretoria()

## Parte 2 – Interpretador de comandos

Este relatório visa fornecer uma explicação detalhada de como o código C está a funcionar. O objetivo do programa é interpretar e executar comandos enviados por linha de comando, dependendo do número e do conteúdo dos argumentos. Este pode exibir o conteúdo de um arquivo, copiar o conteúdo de um arquivo, adicionar conteúdo, contar linhas, apagar arquivos, informar sobre um arquivo e listar diretórios.

O interpretador utiliza funções com ‘fork’ para criar um processo filho que irá interpretar o comando enquanto que o processo pai fica em modo de espera (wait), isto é, enquanto o processo filho não acabar o processo pai não continua. Enquanto o processo filho não terminar a interpretação, o processo pai não indica ao utilizador, através da escrita do caracter “%” na consola, que o programa está pronto para receber o próximo comando.

Isto funciona pois o processo filho não contém nenhuma informação anterior do código, deixando as variaveis presentes como NULL e desta forma conseguimos escolher o que cada processo vai fazer.

Na imagem a baixo, podemos ver que a função fork() é realizada e o pid (Process ID) é guardado, mas apenas o processo pai tem conhecimento do valor, enquanto que o processo filho contem um valor NULL que pode ser utilizado na condição if(!pid) que significa, se pid for um valor inválido.

Desta forma selecionamos o procsso filho para realizar a interpretação enquanto o processo pai aguarda que o processo filho termine e indique ao utilizador que pode introduzir um novo comando.



Figura 10– Fork

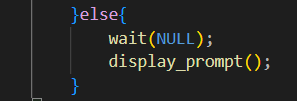


Figura 11– Wait

A leitura de argumentos é realizada através da utilização de agrc e agrv[] que são enviados para a função main diretamente da consola.

Argc: valor inteiro que guarda o número de argumentos incluído o comando.

Argv[]: vetor do tipo char que guarda o comando em argv[0] e os seguintes argumentos no vetor.

Cada comando passado na linha de comandos é verificado usando a função ‘strcmp’ e a quantidade de argumentos com o argc, se a quantidade de argumentos não for respeitada ou se o comando não for reconhecido irá aparecer uma mensagem de erro a mencionar o mesmo problema.

Consoante o argumento passado o codigo irá chamar funções diferentes.

* + **Mostra**- Mostra o conteudo de um arquivo
  + **Copia**- Este comando cria um novo ficheiro cujo nome é ficheiro.copia contendo todo o conteudo do original.
  + Acrescenta- Este comando acrescenta todo o conteudo de um ficheiro de origem para um de destino.
  + Conta- Conta quantas linha tem um determinado ficheiro.
  + Apaga- Apaga um ficheiro com o nome indicado
  + Informa- Este comando apresenta apenas a informação do sistema de ficheiros em relação ao ficheiro indicado.
  + Lista - Este comando apresenta uma lista de todas as pastas e ficheiros existentes na directoria indicada.

## Parte 3 - A

A)

A screenshot of a computer

Description automatically generated

Figura 11 - Criação de um novo disco rígido

A screenshot of a computer

Description automatically generated

Figura 12 - Selecionar o tipo de disco que queremos criar

A screenshot of a computer

Description automatically generated

Figura 13 - Configurações do novo disco rígido

Para que o espaço seja alocado de maneira dinâmica, isto é, ocupado conforme a necessidade, mantemos as caixas em branco. Caso contrário, se escolhermos a opção 'Pre-Allocate Full Size', todo o espaço especificado será imediatamente reservado no disco.

A computer screen shot of white text

Description automatically generated

Figura 14 - Disco criado na máquina virtual

A screenshot of a computer

Description automatically generated

Figura 15 - Criação da partição no disco

A screenshot of a computer program

Description automatically generated

Figura 16 - Partição criada

* n (n de new, para criar uma nova partição)
* p (p de primary, para criar uma partição primária)
* 1 (1 o número da partição a criar)
* Enter (para assumir o início da partição automaticamente)
* Enter (para assumir o fim da partição automaticamente)
* w (w de write, para guardar as alterações na tabela de partições do disco

B)

A screenshot of a computer program

Description automatically generated

Figura 17 - Criação de um novo volume

Na partição designada, foram estabelecidos um Physical Volume (PV) e um Volume Group (VG) usando os comandos "pvcreate" e "vgcreate", respectivamente. Conforme mostrado na figura 7, o comando "pvs" verifica a presença do PV e o comando "vgs" confirma que o VG denominado "vgso" foi criado com sucesso.

A screenshot of a computer

Description automatically generated

Figura 18 - Criação de dois volumes lógicos

Posteriormente, dois Logical Volumes foram configurados. Observa-se na figura 7 que o tamanho do VG é menor que 10GB. Portanto, para criar os LVs com tamanhos aproximadamente iguais, o comando “sudo lvcreate -L 5G -n lvso vgso” foi utilizado para formar o primeiro LV chamado “lvso” com 5GB de espaço, e o segundo LV foi criado com o restante espaço livre do VG usando “sudo lvcreate -l 100%FREE -n lvso2 vgso”. A figura 15 mostra a presença dos dois LVs, onde o “lvso2” possui menos de 5GB.

C)

A screenshot of a computer

Description automatically generated

Figura 19 - Montagem dos volumes em Sistema

Para formar os sistemas de ficheiros, utilizamos o comando "mkfs", aplicando o sistema ext3 a um LV e o ext4 a outro. Durante a execução do comando, especificamos o sistema de arquivos desejado junto ao nome do Logical Volume.

D)

A screenshot of a computer

Description automatically generated

Figura 20 - Edição do ficheiro fstab

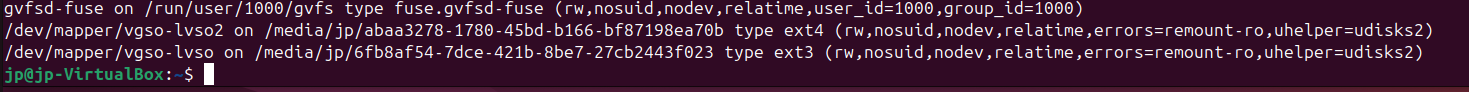


Figura 21 - Identificação dos volumes com o tipo de sistema de ficheiros

Para garantir que o sistema de ficheiros seja persistente após reinicializações, é necessário modificar o ficheiro /etc/fstab, onde o sistema operativo armazena a tabela de sistemas de ficheiros. Isso é feito abrindo o ficheiro num editor de texto. Uma vez aberto, deve-se adicionar as partições criadas à tabela, especificando o dispositivo a ser montado, o ponto de montagem, o tipo de sistema de ficheiros, as opções de montagem, a frequência de backup e se deve ser verificado pelo fsck ou não.

E)

A computer screen shot of a program

Description automatically generated

Figura 22 - Criação do ficheiro com o nº de aluno dos elementos do grupo

A computer screen with white text

Description automatically generated

Figura 23 - Edição das permissões de utilizador

Para criar um ficheiro na diretoria /mnt/ext4, utilizamos inicialmente o comando `touch` para criar o ficheiro e, em seguida, o comando `stat` para verificar as permissões do mesmo. Como o proprietário deve ser o utilizador atual, e não o root, executamos o comando `chown` para modificar o dono do ficheiro. Posteriormente, usamos o comando `chmod` para ajustar as permissões do ficheiro. Neste caso, apenas removemos as permissões de leitura do grupo, já que as permissões para o utilizador atual e para outros já estavam adequadas. Finalizamos executando o comando `stat` novamente para confirmar que as permissões foram ajustadas conforme o esperado.

## Parte 3 – B

A)

A screenshot of a computer

Description automatically generated

Figura 24 - Bloco de início dos ficheiros regulares

Como o sistema de ficheiros começa no setor 0 conforme a figura 14, não é necessário calcular um offset para a partição.

B)

A black background with white text

Description automatically generated

Figura 25 - Conteúdo do ficheiro

# Conclusão

A realização deste trabalho prático foi muito valorizada por todos os membros do grupo, pois não só possibilitou a aplicação dos conhecimentos teóricos adquiridos em diversas áreas durante o semestre na disciplina de Sistemas Operativos, mas também permitiu um aprofundamento significativo de vários conceitos anteriormente não completamente dominados. Isso ocorreu especialmente devido aos desafios encontrados, principalmente relacionados à gestão de processos e às funções de chamadas ao sistema, que exigiram pesquisas constantes, resultando numa evolução notável dos nossos conhecimentos em áreas como Linux, linguagem C e máquinas virtuais. Essa experiência ampliou a nossa compreensão sobre como certos processos são geridos pelo sistema operativo, concluindo o projeto gratos pelo enriquecimento proporcionado.