Licenciatura em Engenharia e Gestão de Sistemas de Informação

Sistemas Operativos

Ano letivo 2023/2024

Enunciado do Trabalho Prático

20 outubro/2023

Introdução

Este documento descreve o trabalho prático a realizar no âmbito da unidade curricular de Sistemas Operativos. A realização deste trabalho prático pretende promover a aquisição de conhecimentos de gestão de ficheiros, gestão de processos em Linux e de programação baseada em chamadas ao sistema, tentando assim corresponder a alguns dos objetivos de aprendizagem da unidade curricular. Este trabalho prático será também a principal forma de avaliação dessas mesmas competências e um contexto de aprendizagem capaz de promover a consolidação dos conceitos mais teóricos abordados nas aulas. É um pressuposto deste enunciado que os alunos disponham de competências adequadas em programação na linguagem C e trabalho em equipa.

Descrição do trabalho prático

O objetivo deste trabalho prático é desenvolver um comando/programa chamado `mycmd` que permite obter informação sobre os processos ativos no sistema (**sem recurso** aos comandos do Linux) e a execução de comandos Linux com redireccionamento de entrada e saída.

A Tabela 1 descreve a sintaxe e semântica dos comandos a desenvolver. No caso de o comando submetido não ser válido, o programa `mycmd` deverá apresentar uma mensagem de erro, explicativa do problema que ocorreu.

Nº	Comando[1]	Descrição
1	`mycmd top`	Apresenta uma linha com a seguinte informação sobre o sistema: carga média do CPU (*load average*) nos últimos 1, 5 e 15 minutos; número total de processos e número total de processos no estado de *running*. Apresenta ainda a informação sobre os processos correntemente ativos no sistema, nomeadamente o `pid`, o estado, a linha de comando e o *username*. O *output* deste comando deve ser refrescado em intervalos de 10s até receber o *input* 'q'. Em cada ciclo deve apresentar os processos no estado *running*, ordenados por ordem crescente do PID, no máximo de 20 processos. Caso não existam 20 processos no estado *running*, deverá completar a lista com os restantes processos, também ordenados por ordem crescente do PID.
2	`mycmd cmd arg1 ... argn` (ex: `mycmd ls -l`)	Executa o comando Linux *cmd.*
3	`mycmd cmd arg1 ... argn ">" file` (ex: `mycmd ls -l ">" ls.txt`)	Executa o comando Linux *cmd*; redireciona a saída (*output*) do comando para o ficheiro *file*.
4	`mycmd cmd arg1 ... argn "<" file` (ex: `mycmd grep TEXTO "<" exemplo.txt`)	Executa o comando Linux *cmd*; redireciona a entrada (*input*) do comando para o ficheiro *file*
5	`mycmd cmd1 arg1 ... argn "|" cmd2 arg1 ... argn` (ex: `mycmd ps -A "|" grep 12345`)	Executa os comandos Linux *cmd1* e *cmd2*; redireciona a saída do comando *cmd1* para a entrada do comando *cmd2*.

Tabela 1 Comandos a desenvolver

[1] No caso dos comandos 3, 4 e 5, os caracteres >, < e | devem estar entre aspas, caso contrário, a *bash* irá assumir o redireccionamento.

Trabalho a realizar

Cada grupo deverá desenvolver o programa `mycmd` de acordo com a sintaxe e semântica apresentadas. O programa `mycmd` deverá ser desenvolvido na linguagem C, em Linux, numa instalação local, tendo em contas as seguintes especificações técnicas:

- A informação sobre o desempenho do sistema e dos processos deverá ser obtida através do sistema de ficheiros *proc,* comummente montado (*mounted*) na diretoria `/proc`. Este sistema de ficheiros disponibiliza um interface de acesso à informação das estruturas de dados do *kernel* baseada na leitura de ficheiros. Pode consultar a estrutura deste sistema de ficheiros recorrendo ao manual do Linux com o comando `man 5 proc`. Em particular, este sistema de ficheiros é constituído por um conjunto de diretorias identificadas pelos PIDs dos processos que disponibilizam informação sobre os processos.
- O acesso aos dados das diferentes diretorias poderá ser feito recorrendo às funções da biblioteca C, nomeadamente:

  ```
  DIR *opendir(const char *name)
  struct dirent *readdir(DIR *dirp)
  ```

- O acesso aos ficheiros deverá ser feito exclusivamente através de chamadas ao sistema do sistema de ficheiros, nomeadamente:

  ```
  int open(const char *path, int oflag [,mode)
  ssize_t read(int fildes, void *buf, size_t nbyte)
  int close(int fildes)
  ....
  ```

  Ver Anexos para algumas notas sobre estas chamadas ao sistema `dup2()`.
- O redireccionamento do *input* e *output* dos comandos para ficheiros será implementado recorrendo à chamada ao sistema:

  ```
  int dup2(int oldfd, int newfd)
  ```

  Ver Anexos para algumas notas sobre a chamada ao sistema `dup2()`.
- O redireccionamento do *output* de um comando para o *input* de outro comando será implementado recorrendo a *pipes* anónimos:

  ```
  int pipe(int pipefd[2])
  ```

  Ver Anexos para algumas notas sobre a chamada ao sistema `pipe()`.

Instruções para a execução do trabalho

O trabalho será realizado em grupo de **3 elementos**. A constituição de cada grupo deverá ser comunicada ao docente de cada turno **até ao final do dia 27/10/2023**.

Planeamento do trabalho

O trabalho encontra-se estruturado de forma a corresponder a dois momentos de avaliação.

Momento 1 Semana de 04/12	Neste momento, o grupo deverá apresentar oralmente uma proposta de solução de alto nível que descreva uma visão sobre as principais questões a considerar no desenvolvimento deste programa e um primeiro esboço da implementação dos comandos a considerar.
Momento 2 Entrega final: 07/01/2024 Demonstração final: Semana de 08/01 em horário a definir	Este momento corresponde à entrega final de todos os elementos de avaliação (**código desenvolvido** e **relatório final**) e à **defesa individual do projeto**. Todos os elementos do grupo devem estar presentes para defender o trabalho. Esta defesa consistirá numa demonstração do programa desenvolvido. Os alunos vão defender individualmente as suas opções perante a equipa docente. Será disponibilizado oportunamente o escalonamento das demonstrações que decorrem na semana de 08/01/2024

O relatório final deverá apresentar os objetivos do trabalho, as principais questões a considerar e os fundamentos teóricos/tecnológicos subjacentes, a descrição da implementação dos comando, um manual dos comandos e as conclusões.

Avaliação

A avaliação final terá em conta o trabalho realizado em cada momento de avaliação, o código final, o relatório final, o cumprimento dos prazos e a defesa/demonstração do trabalho realizada individualmente por cada aluno, contribuindo cada um destes elementos de igual forma para a classificação de cada momento de avaliação do projeto. Nos casos em que a defesa individual não seja satisfatória (o que implica a atribuição de uma nota inferior em relação à classificação do trabalho ou a reprovação do aluno), e se o aluno o pretender, haverá uma segunda chamada de defesa do contributo para o projeto (a agendar conforme disponibilidade do aluno e equipa docente).

Na entrega de código deve incluir-se todos os ficheiros C. Os ficheiros de código devem fazer parte de um único ficheiro em formato de texto correspondente à concatenação dos vários ficheiros `.c` que sejam desenvolvidos (não deve ser incluído qualquer código que seja disponibilizado pela equipa docente). As entregas do código e do relatório deverão ser realizadas em formato eletrónico na página da UC no *elearning* no link correspondente.

A avaliação terá em conta a abrangência do trabalho realizado, sendo definidas as seguintes notas máximas:

Trabalho realizado	Nota máxima a atribuir
Implementação dos comandos 1 (recorrendo às chamadas ao sistema do sistema de ficheiros), 2, 3, 4 e 5, com deteção de erros.	20
Implementação dos comandos 1 (recorrendo às chamadas ao sistema do sistema de ficheiros), 2, 3 ou 4 e 5, com deteção de erros.	18
Implementação dos comandos 1 (recorrendo às chamadas ao sistema do sistema de ficheiros), 2, 3 e 4, com deteção de erros.	16
Implementação dos comandos 1 (recorrendo às chamadas ao sistema do sistema de ficheiros), 2, 3 ou 4, com deteção de erros.	14
Implementação dos comandos 1 (recorrendo às chamadas ao sistema do sistema de ficheiros) e 2	12
Implementação dos comandos 1, 2, 3, 4 e 5	16
Implementação dos comandos 1, 2, 3 ou 4 e 5	14
Implementação dos comandos 1, 2, 3 e 4	12
Implementação dos comandos 1, 2, 3 ou 4	10
Implementação dos comandos 1 e 2 ou 3 e 4 ou 5	8

O trabalho realizado até ao **momento 1** será avaliado de 0 a 3. As classificações de 0 e 1 no momento 1 de avaliação, implicam decrementar a nota máxima em 2 valores.

Autoria

A autoria do projeto será avaliada com recurso à ferramenta *SafeAssign* do sistema *Blackboard* ou equivalente. Uma vez que todos os grupos estão a fazer o mesmo projeto é normal que ocorra alguma partilha de conhecimento entre os grupos. No entanto, quaisquer interações entre grupos devem ter sempre em conta o princípio fundamental de que cada grupo tem de chegar de forma autónoma e independente à sua solução para o problema. Para evitar problemas com a autoria dos trabalhos, sugere-se que os alunos tenham em conta os seguintes referenciais relativamente ao que é ou não permitido:

- Não prestar nem receber qualquer ajuda que seja específica do problema proposto, como por exemplo a forma de implementar determinada parte do trabalho.
- NUNCA partilhar código ou relatório entre os grupos, já que trabalhos que apresentem semelhanças de implementação ou no relatório serão encarados como uma tentativa de fraude e serão anulados.
- Os grupos podem discutir entre si:

    - Os objetivos e a interpretação do problema.
    - A utilização em geral (não especificamente no contexto deste trabalho) das tecnologias necessárias para a realização do projeto.
- Qualquer utilização de código retirado de outras fontes deve ser explicitamente assinalada e justificada.
- A utilização de código obtido diretamente através do `chatGPT` será encarada como uma tentativa de fraude e implicará a anulação do trabalho. Isto não quer dizer que os grupos estejam impedidos de interagir com o chatGPT para clarificar alguma dúvida. No entanto, não será permitido a reutilização de código gerado pelo chatGPT.

Anexo I - Argumentos de uma programa

Quando um programa é executado, é passada ao processo uma lista de argumentos com tamanho variável. Num programa em C essa lista é passada como um parâmetro da função `main()`.

A função `main()` é declarada da seguinte forma:

```
int main(int argc, char *argv[]) {
....
}
```

O primeiro argumento, `argc`, especifica o número de argumentos. Este valor é sempre igual ou maior do que 1, uma vez que é sempre considerado pelo menos um argumento: o nome pelo qual o programa é invocado.

O segundo argumento, `argv`, é um *array* de apontadores para *strings*. Cada apontador no *array* `argv` aponta para o nome de um argumento.

Foi disponibilizado na página da UC uma função C (a função `parse()`) que recebe os argumentos da linha de comando a desenvolver e devolve uma estrutura que identifica os argumentos de cada um dos comandos a executar. Esta função pode ser utilizada no desenvolvimento do trabalho da forma que acharem mais adequada.

Anexo II – Chamadas ao sistema - Retorno de erros

Em caso de erro, as chamadas ao sistema retornam normalmente -1 (verificar no manual).

Os erros são classificados por inteiros positivos. Estes valores são armazenados na variável global `errno`, definida em `errno.h`. A função `perror()` (ver `man 3 perror`) imprime uma mensagem explicativa do erro correntemente armazenado em `errno`.

Anexo III – As chamadas ao sistema `open()`, `read()` e `close()`

De entre as chamadas ao sistema do sistema de ficheiros, as chamadas `open()`, `read()` e `close()` assumem um maior importância na realização deste trabalho.

Descritor de ficheiro

- Representação abstrata de um ficheiro utilizada para operar sobre o mesmo
- Faz parte da interface POSIX
- Representado por um inteiro não negativo
- Pode também servir para representar outros recursos de *Input*/*Output* como *pipes*, *sockets*, dispositivos de entrada ou saída, e.g., teclado

Descritores standard (podem ser redefinidos – ver Anexo III e Anexo IV):

Valor inteiro	Nome
0	Standard input
1	Standard output
2	Standard error

Tabela de descritores de ficheiros (TDF)

- Existe uma tabela por processo
- Guarda descritores de ficheiros acertos

Tabela de ficheiro (TF)

- Guarda modo de abertura e posição de leitura/escrita de cada descritor

I-node

- Guarda metadados/atributos do ficheiros (p.ex: nome ficheiro, tamanho, ...)
- Guarda localização dos dados no recurso físico de armazenamento

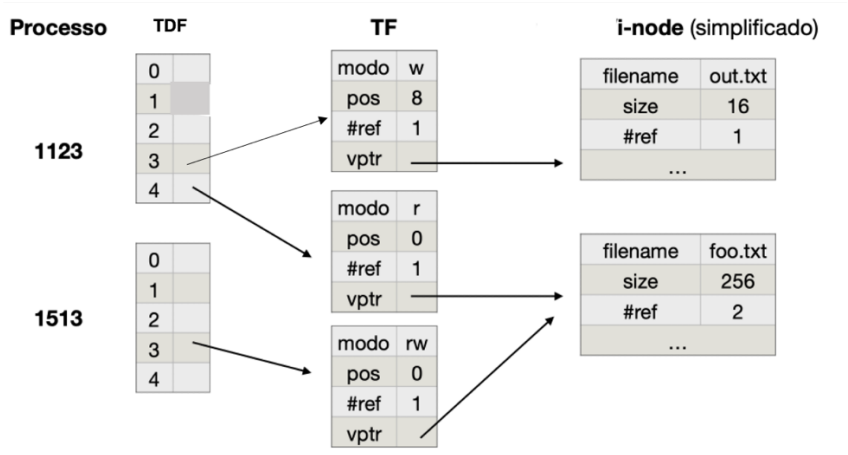

Figura 1 Descritores de ficheiros, TDF, TF e i-node

Por exemplo, na Figura 1 o processo 1123 tem dois ficheiros abertos, `out.txt` e `foo.txt`, cujos descritores são o 3 e 4 respetivamente. O processo 1513 tem um ficheiro aberto, `foo.txt`, cujo descritor é o 3.

Chamada ao sistema **`open()`** (ver `man 2 open`)

```
int open(const char *path, int oflag [,mode)
```

A chamada ao sistema `open()` retorna um descritor que será usado nos acessos ao ficheiro posteriores, por exemplo na leitura de um ficheiro através da chamada ao sistema `read()`.

Chamada ao sistema **`read()`** (ver `man 2 read`)

```
ssize_t read(int fildes, void *buf, size_t nbyte)
```

A chamada ao sistema `read()` tenta ler até `nbyte` *bytes* do descritor de ficheiro `fd` para o *buffer* que inicia no apontador `buf`. Se tiver sucesso, retorna o número de bytes efetivamente lidos.

A cada operação de leitura (e também escrita) efetuada sobre o mesmo descritor de ficheiro, a posição a ler (ou escrever) é atualizada consoante o número de *bytes* efetivamente lidos (ou escritos).

Se a posição for o final do ficheiro, a chamada `read()` retorna `EOF` (*end-of-file*).

Chamada ao sistema **`close()`** (ver `man 2 close`)

```
int close(int fildes)
```

A chamada ao sistema `close()` apaga o descritor da tabela de descritores do processo.

Anexo IV – Redireccionamento de I/O

Chamada ao sistema `dup2()` (ver `man 2 dup2`)

```
int dup2(int oldfd, int newfd)
```

A chamada ao sistema `dup2()` cria uma cópia do descritor `fd1` no descritor `fd2`, especificado pelo utilizador. Devolve valor de `fd2` ou erro. Se o descritor `fd2` estiver aberto é chamado um `close()` implícito. Mantém modo e posição do descritor original.

Considere o exemplo representado na Figura 2.

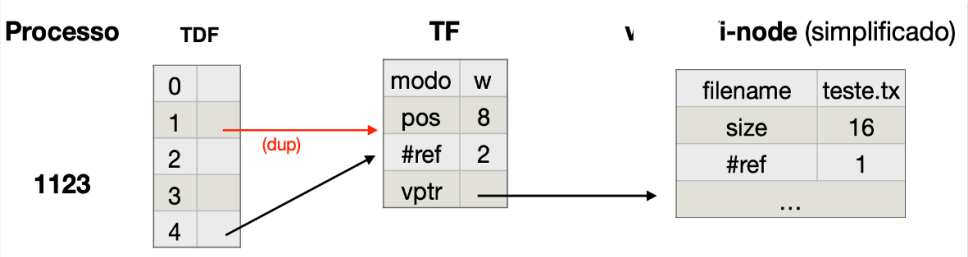

Figura 2 Exemplo chamada ao sistema `dup2()`

O processo 1123 abre o ficheiro teste.txt cujo descritor é o 4. Após a invocação da chamada ao sistema `dup2(4,1)`, o descritor 1 (*standard output*) aponta para o ficheiro `teste.txt`. Qualquer invocação da função `printf()`, por exemplo, a qual envia dados para o *standard output*, enviará dados para o ficheiro `teste.txt`.

Material de apoio adicional

- https://www.gnu.org/software/libc/manual/html_node/Duplicating-Descriptors.html

Anexo V – *pipes* anónimos

Um *pipe* anónimo é um buffer em memória, com a sincronização entre processos produtores (escritas) e processos consumidores (leituras) gerida pelo kernel. O processo consumidor bloqueia na leitura se não há nada para ler. O processo produtor bloqueia na escrita se não há espaço para escrever. A semântica de comunicação é tipo FIFO, apenas num só sentido. Permitem combinar programas sem os modificar, por exemplo *shell*: `ls | more`.

Chamada ao sistema `pipe()` (ver `man 2 dup2`)

```
int pipe(int pipefd[2])
```

A chamada ao sistema `pipe()` recebe um *array* de 2 inteiros no qual aloca 2 descritores de ficheiros. Os dados escritos para o descritor `fildes[1]` podem ser lidos através do descritor `fildes[0]`.

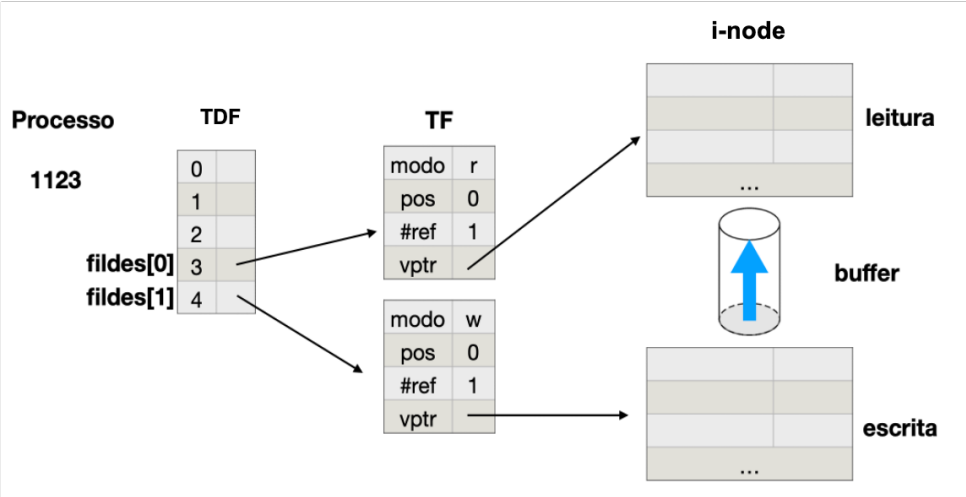

Figura 3 Pipes anónimos (exemplo)

No exemplo da Figura 3, o processo 1123 criou um *pipe* anónimo. Os dados enviados para o descritor 4 (`fildes[1]`), podem ser lidos no descritor 3 (`fildes[0]`).

Algumas considerações

- A leitura de um *pipe* apenas devolve EOF (*end-of-file*) se todos os descritores de escrita estiverem fechados
  - Processos que lêem de um *pipe* devem fechar o descritor de escrita e vice-versa.
  - Descritores abertos que deveriam estar fechados podem dar origem a *deadlocks*.

Material de apoio adicional

- https://www.gnu.org/software/libc/manual/html_node/Pipes-and-FIFOs.html#Pipes-and-FIFOs