

Consola, Ficheiros e Directórios

Jorge Silva

MIEIC / FEUP

Objectivos

No final desta aula, os estudantes deverão ser capazes de:

- Explicar o que é um descritor de um ficheiro e quais são as principais estruturas de dados, usadas pelo sistema de ficheiros do UNIX/LINUX
- Identificar os 3 descritores *standard*
- Manipular algumas características da consola (ex: eco)
- Manipular ficheiros (criar, abrir, ler, escrever, destruir, ...), usando chamadas ao sistema
- Usar as chamadas `dup()` e `dup2()` e explicar a sua utilização no redireccionamento de entradas/saídas
- Usar as principais chamadas ao sistema relativas à manipulação de directórios (criar, listar os ficheiros/sub-dir.s, obter as propriedades de um ficheiro/sub-dir., ...)
- Explicar o conceito de *hard link* e *symbolic link* entre ficheiros

Desafios

Escrever programas para:

1.

Ler uma *password* sem ecoar os caracteres escritos pelo utilizador

> read_password

2.

Copiar um ficheiro para outro ou mostrá-lo o ecrã, dependendo do nº de argumentos da linha de comandos

> copy source // mostra no écran

> copy source destination // copia p/outro ficheiro

3.

Listar os ficheiros regulares e sub-directórios de um directório e ...

> listdir dirname

Ficheiros

As chamadas ao sistema relacionadas com ficheiros permitem manipular ficheiros simples, directórios e ficheiros especiais, incluindo:

- ficheiros em disco
- terminais
- impressoras
- facilidades para intercomunicação entre processos, tais como *pipes* e *sockets*.

Descritores de ficheiros

Para o *kernel* todos os ficheiros abertos são referidos através de descritores.

Quando se cria um ficheiro ou se abre um ficheiro já existente, o *kernel* retorna um descritor ao processo que criou ou abriu o ficheiro. Este descritor é um dos argumentos das chamadas que permitem ler ou escrever no ficheiro.

Um descritor é um número inteiro, não-negativo, geralmente pequeno. Os descritores podem tomar valores entre 0 e `OPENMAX`.

Por convenção, as *shells* de Unix associam os 3 primeiros descritores a ficheiros especiais: 0 - *standard input*; 1- *standard output*; 2 - *standard error*

Estes descritores estão definidos em `unistd.h` através de constantes : `STDIN_FILENO`, `STDOUT_FILENO` e `STDERR_FILENO`

Por exemplo, a função `printf()` escreve sempre usando o descritor 1 e a função `scanf()` lê sempre usando o descritor 0.

Quando se fecha um ficheiro, o descritor correspondente é libertado e pode ser reutilizado quando se abre um novo ficheiro.

Um ficheiro pode ser aberto várias vezes e por isso pode ter vários descritores a ele associados.

Descritores de ficheiros

Cada descritor de ficheiro tem um conjunto de propriedades associadas:

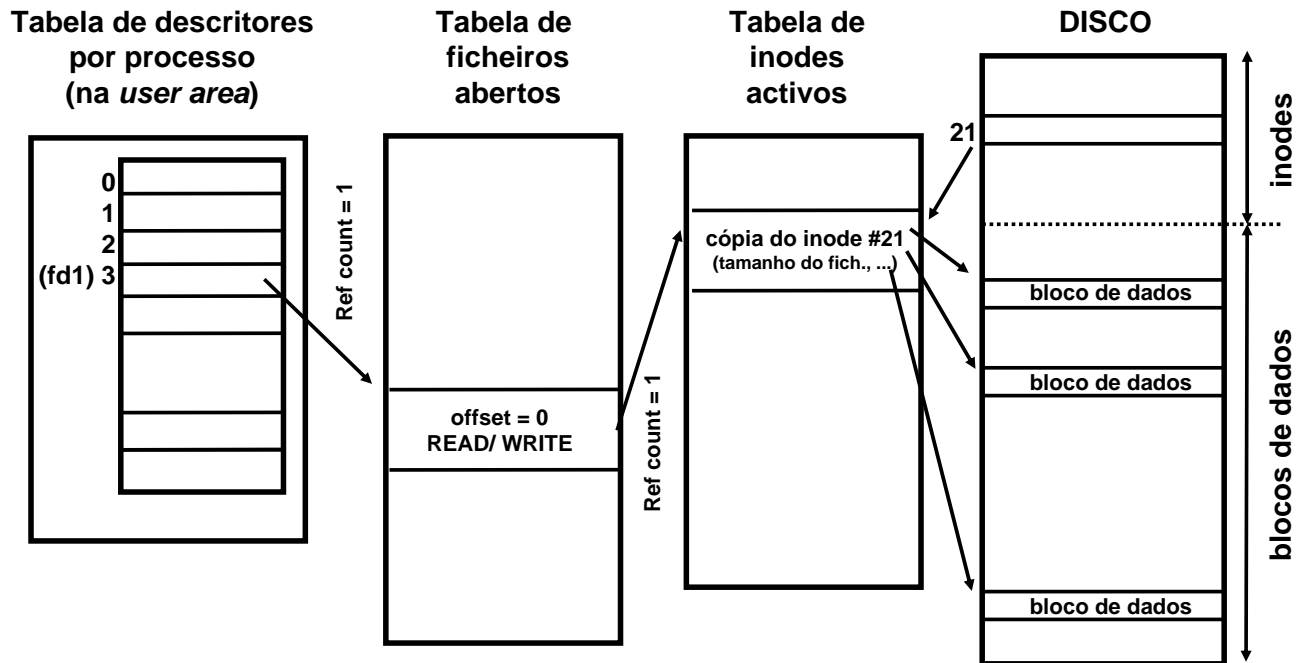
- um apontador (cursor) de ficheiro que indica a posição do ficheiro onde será executada a próxima operação de leitura/escrita
 - » colocado a 0 (zero) quando o descritor é criado
 - » avança automaticamente após cada operação de leitura/escrita
- uma *flag* que indica se o descritor deve ser automaticamente fechado se o processo invocar uma das funções `exec()`
- uma *flag* que indica se o que se escreve para o ficheiro deve ser acrescentado no fim do ficheiro

Existem outras propriedades que só se aplicam a ficheiros especiais, como *pipes* e *sockets*:

- uma *flag* que indica se um processo deve bloquear se tentar ler de um ficheiro quando ele está vazio.
- um número que indica o identificador de um processo ou de um grupo de processos a quem deve ser enviado o signal `SIGIO` se passarem a existir dados no ficheiro.

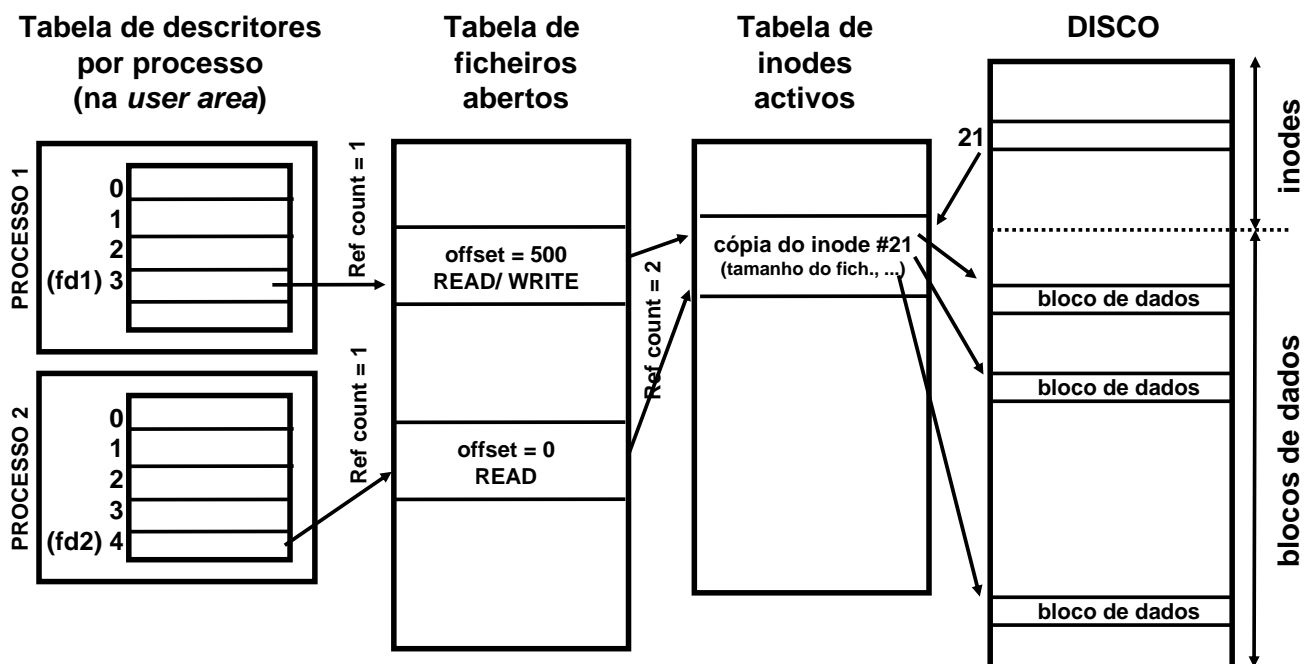
ESTRUTURAS DE DADOS DO S.O. RELATIVAS AO SISTEMA DE FICHEIROS

```
fd1=open( "/usr/John/doc1.txt",O_RDWR)
```



ESTRUTURAS DE DADOS DO S.O. RELATIVAS AO SISTEMA DE FICHEIROS

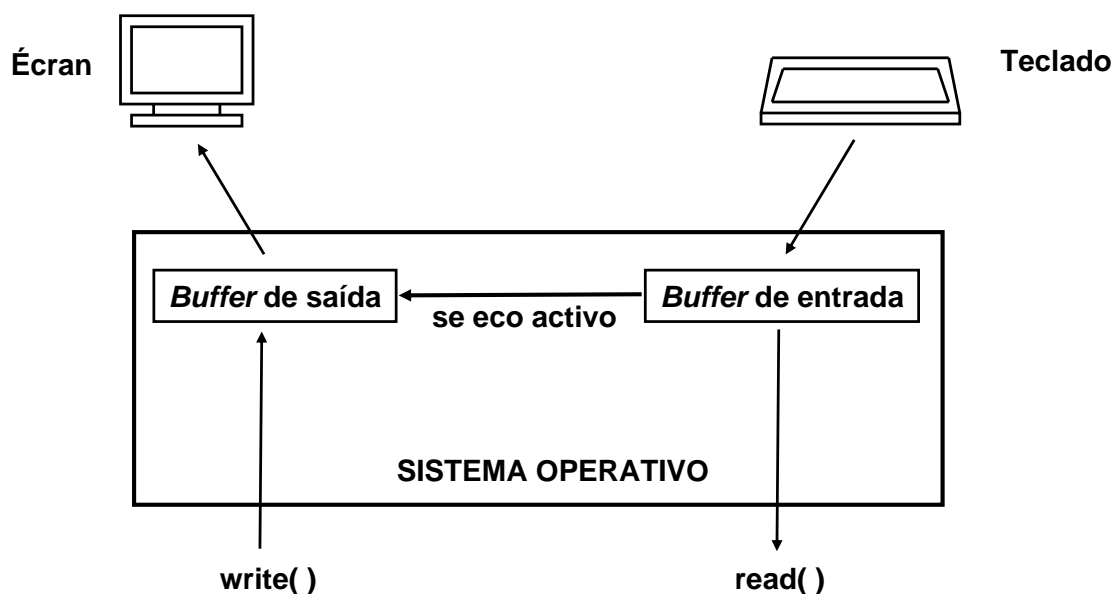
2 processos diferentes abrem o mesmo ficheiro



Consola

- A consola (teclado + ecrã) é vista pela generalidade dos S.O.'s como um ou mais ficheiros onde se pode ler ou escrever texto.
- Esses ficheiros são normalmente abertos pela rotina de *C startup*.
- A biblioteca standard de C inclui diversas funções de leitura e escrita directa nesses ficheiros:
 - » `printf()`, `scanf()`, `getchar()`, `putchar()`, ...
- Também é possível aceder àqueles periféricos através de serviços dos S.O.'s
 - » o Unix não define serviços especiais de leitura e escrita na consola
 - » deverão usar-se os serviços genéricos de leitura e escrita em ficheiros

Consola



Consola

Modos de funcionamento da consola em Unix:

- **modo canónico (*cooked*)**
 - existe uma série de caracteres especiais de entrada que são processados pela consola e não são transmitidos ao programa que está a ler
 - » ex: `ctrl-U`, `ctrl-H`, `ctrl-S`, `ctrl-Q`, ...
 - » muitos destes caracteres são alteráveis programaticamente
 - a entrada só é passada ao programa quando se tecla `<Return>`
- **modo primário (*raw*)**
 - não há qualquer processamento prévio dos caracteres teclados
 - eles são passados um a um ao programa

Consola

Alteração das características da consola em Unix:

```
#include <termios.h>

int tcgetattr(int filedes, struct termios *term_ptr);
int tcsetattr(int filedes, int opt, const struct termios *term_ptr);
```

`tcgetattr()` - preenche uma estrutura `termios` cujo end^o é passado em `term_ptr` com as características do componente da consola cujo descritor é `filedes`

`tcsetattr()` - modifica as características da componente da consola cujo descritor é `filedes`, com os valores previamente colocados em `termios` cujo end^o é passado em `term_ptr`

`opt` indica quando a modificação irá ocorrer:

- `TCSANOW` -> imediatamente
- `TCSADRAIN` -> após *buffer* de saída se esgotar
- `TCSAFLUSH` -> após *buffer* de saída se esgotar; além disso, esvazia *buffer* de entrada

Consola

```
struct termios {
    tcflag_t c_iflag; /* input flags */
    tcflag_t c_oflag; /* output flags */
    tcflag_t c_cflag; /* control flags */
    tcflag_t c_lflag; /* local flags */
    cc_t c_cc[NCCS]; /* control characters */
}
```

`c_iflag, c_oflag, c_cflag, c_lflag`:
campos constituídos por *flags* de 1 ou mais *bits*
que permitem controlar as características da consola

`c_cc[]`:
array onde se definem os caracteres especiais que são processados pela consola
quando esta estiver a funcionar em modo canónico
ex:
`mytermios.c_cc[VERASE]=8; /* 8 = código ASCII de <ctrl-H> */`

O comando da *shell* `stty -a` permite ver os *settings* da estrutura `termios`

Consola

Frequentemente, pretende-se apenas activar ou desactivar determinadas *flag* dos campos de `termios` sem alterar as outras

Exemplo:

```
struct termios oldterm, newterm;

...

tcgetattr(STDIN_FILENO, &oldterm);
newterm=oldterm;
newterm.c_lflag &= ~(ECHO | ECHOE | ECHOK | ECHONL | ICANON);
tcsetattr(STDIN_FILENO, TCSAFLUSH, &newterm);
... /* executar operações usando a "nova consola" */
tcsetattr(STDIN_FILENO, TCSANOW, &oldterm);
...
```

Consola

- Se acontecer algum erro que leve a que um programa que alterou as características da consola termine imprevistamente, pode acontecer que ela fique num estado que impossibilite a interacção com o utilizador.
- Para tentar repôr o estado "normal" existem várias alternativas:
 - 1) `stty sane` seguido de <return> ou <ctrl-J>
 - 2) `stty -g >save_stty` seguido de <return> ou <ctrl-J>
... (correr o programa)
`stty $(cat save_stty)` seguido de <return> ou <ctrl-J>
 - 3) ou ...
fechar a consola actual e abrir outra ... 😊

Criação/abertura de ficheiros

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int oflag, ... /*, mode_t mode */);

Retorna: descritor do ficheiro se OK, -1 se erro
```

`pathname` = nome do ficheiro

`oflag` = combinação de várias *flags* de abertura

<code>O_RDONLY</code>	- abertura só para leitura	<- só uma destas 3
<code>O_WRONLY</code>	- abertura só para escrita	
<code>O_RDWR</code>	- abertura para leitura e escrita	
<code>O_APPEND</code>	- p/ acrescentar no fim do ficheiro	
<code>O_CREAT</code>	- p/ criar o ficheiro se ele não existir; <u>requer</u> <u>mode</u>	
<code>O_EXCL</code>	- origina erro se o ficheiro existir e <code>O_CREAT</code> estiver activada	
<code>O_TRUNC</code>	- se o ficheiro existir fica com o comprimento 0	
<code>O_SYNC</code>	- só retorna depois de os dados terem sido fisicamente escritos	
...	no ficheiro	

mode = permissões associadas ao ficheiro

- só devem ser indicadas quando se cria um novo ficheiro
- pode ser o *OR bit a bit* (|) de várias das seguintes constantes:

S_IRUSR - *user read*
 S_IWUSR - *user write*
 S_IXUSR - *user execute*
 S_IRGRP - *group read*
 S_IWGRP - *group write*
 S_IXGRP - *group execute*
 S_IROTH - *others read*
 S_IWOTH - *others write*
 S_IXOTH - *others execute*

Alternativa:

	owner	group	other
	rwX	rwX	rwX
	111	101	000
→	7	5	0
mode (em octal)	#define MODE 0750		

Nota:

- as permissões efectivas podem não ser exactamente as especificadas, consoante o valor da "*file creation mask*" (especificada c/ a chamada `umask`)
- o valor por omissão desta máscara é frequentemente 022 (octal) o que significa anular as permissões de escrita excepto para o *owner*

Duplicação de um descritor

Pode ser feita c/ as funções *dup* ou *dup2*.

```
# include <unistd.h>

int dup (int filedes);
int dup2 (int filedes, int filedes2);

Retornam: novo descritor se OK, -1 se houve erro
```

- **dup**
 - procura o descritor livre c/ o número mais baixo e põe-no a apontar p/ o mesmo ficheiro que *filedes*.
- **dup2**
 - fecha *filedes2* se ele estiver actualmente aberto e põe *filedes2* a apontar p/ o mesmo ficheiro que *filedes*;
 - se *filedes=filedes2*, retorna *filedes2* sem fechá-lo.
 - exemplo:

```
dup2 (fd, STDIN_FILENO)
```

 redirecciona a entrada standard (teclado) para o ficheiro cujo descritor é *fd*.

`dup2 (fd1 , fd2)`

Tabela de descritores
por processo
(na *user area*)

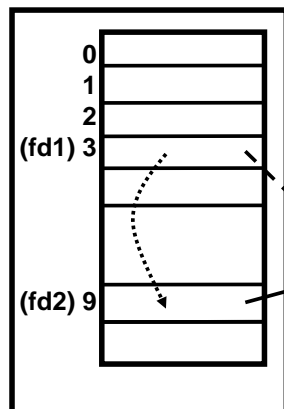


Tabela de
ficheiros
abertos

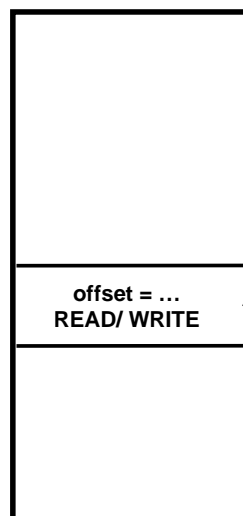
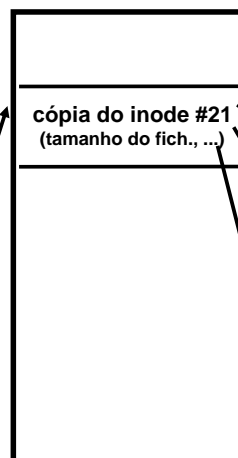
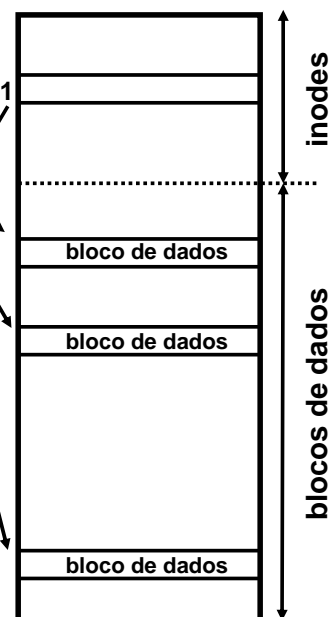


Tabela de
inodes
activos



DISCO



Leitura de um ficheiro

```
#include <unistd.h>
```

```
ssize_t read(int filedes, void *buff, size_t nbytes);
```

Retorna: o nº de bytes lidos, 0 se fim do ficheiro, -1 se erro

`filedes` = descritor do ficheiro

`buff` = apontador p/o *buffer* onde serão colocados os valores lidos

`nbytes` = nº de *bytes* a ler

Esta função não tem nenhuma das capacidades de formatação de `scanf()`.

Escrita num ficheiro

```
#include <unistd.h>
```

```
ssize_t write(int filedes, const void *buff, size_t nbytes);
```

Retorna: o nº de bytes escritos, -1 se erro

`filedes` = descritor do ficheiro

`buff` = apontador p/o *buffer* onde devem ser colocados os valores a escrever

`nbytes` = nº de *bytes* a escrever

Se a *flag* `O_APPEND` tiver sido especificada ao abrir o ficheiro o apontador do ficheiro é posto a apontar para o fim do ficheiro antes de ser efectuada a operação de escrita.

Esta função não tem nenhuma das capacidades de formatação de `printf()`.

Deslocamento do apontador do ficheiro

```
#include <sys/types.h>
```

```
#include <unistd.h>
```

```
off_t lseek(int filedes, off_t offset, int whence);
```

Retorna: novo valor do apontador se OK, -1 se erro

`filedes` = descritor do ficheiro

`offset` = deslocamento (pode ser positivo ou negativo)

`whence` = a interpretação dada ao `offset` depende do valor deste argumento:

- `SEEK_SET` - o `offset` é contado a partir do início do ficheiro
- `SEEK_CUR` - o `offset` é contado a partir da posição actual do apontador
- `SEEK_END` - o `offset` é contado a partir do fim do ficheiro

Para determinar a posição actual do apontador do ficheiro fazer

```
curr_pos = lseek(fd,0,SEEK_CUR)
```

Fecho de um ficheiro

```
#include <unistd.h>

int close(int filedес);

Retorna: 0 se OK, -1 se erro
```

`filedes` = descritor do ficheiro

Fechar um descritor que já tinha sido fechado resulta num erro.

Quando um processo termina,
todos os ficheiros abertos são automaticamente fechados pelo *kernel*.

Se `filedes` for o último descritor associado a um ficheiro aberto
o *kernel* liberta os recursos associados a esse ficheiro
quando se invoca `close()`.

Apagamento de um ficheiro

```
#include <unistd.h>

int unlink(const char *pathname);

Retorna: 0 se OK, -1 se erro
```

`pathname` = nome do ficheiro

Para se apagar um ficheiro é preciso ter permissão de escrita e execução
no directório onde o ficheiro se encontra.

O ficheiro só será, de facto, apagado

- quando for fechado, caso esteja aberto por ocasião da chamada *unlink*
- quando a contagem do nº de *links* do ficheiro atingir o valor 0.

Outras chamadas

umask – modifica a máscara (parâmetro `mode` da chamada `open`) de criação de ficheiros e directórios

stat, fstat, lstat

- retornam uma `struct` com diversas informações acerca de um ficheiro
 - » tipo, permissões, tamanho, nº de *links*, hora da última modificação, ...)
- existe um conjunto de macros (`S_ISREG()`, `S_ISDIR()`, ...) que permitem determinar qual o tipo de ficheiro, a partir de um campo dessa `struct`

mkdir - cria um novo directório

rmdir - apaga um directório

opendir / closedir - abre / fecha um directório

readdir - lê a entrada seguinte do directório e avança automaticam.

rewinddir - faz com que a próxima leitura seja a da 1ª entrada

getcwd - obtém o nome directório corrente

chdir - muda o directório corrente

Outras chamadas (biblioteca de C)

`FILE *fp;`

`fp = fopen(filename, mode)`

`fprintf`

`fscanf`

`fputs`

`fgets`

`fread`

`fwrite`

`feof`

`ferror`

`fclose`

`fdopen` // get a `FILE*` from a file descriptor

`fileno` // get file descriptor from `FILE*`