



Programar com Ficheiros

Sistemas Operativos – DEI - IST

1



Todos Já Programaram com Ficheiros

- Programar com ficheiros é dos aspetos mais básicos da aprendizagem de programação
- Nas linguagens de alto nível está incorporado no modelo

```
f = open("tests.txt", "r")  
print(f.read())
```

Sistemas Operativos – DEI - IST

2

1



Programar com ficheiros em C

- Em linguagens mais antigas é uma funcionalidade proporcionada por bibliotecas que têm de estar explicitamente incluídas no programa

```
FILE *fp;
```

Ponteiro para estrutura que representa o ficheiro aberto

```
fp = fopen("tests.txt", "r");
```

Modo de abertura do ficheiro. Neste caso abrimos o ficheiro em modo de leitura

Sistemas Operativos – DE1 - IST

3



Exemplo 1

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;

    fp = fopen("teste.txt", "r");
    if (fp == NULL) {
        printf("teste.txt: No such file or directory\n");
        exit(1);
    }

    return 0;
}
```

Fundamental incluir a biblioteca

Se não conseguir abrir, fp fica igual a NULL

Sistemas Operativos – DE1 - IST

4

2



Exemplo 2

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *myfile; int i;
    float mydata[100];
    myfile = fopen("info.dat", "r");
    if (myfile== NULL) {
        perror("info.dat");
        exit(1);
    }
    for (i=0;i<100;i++)
        fscanf(myfile,"%f",&mydata[i]);

    fclose(myfile);
    return 0;
}
```

Lê um conjunto de 100 números reais (float) guardados num ficheiro

Sistemas Operativos – DE1 - IST

5



Funções Sistema

- Podemos dividir as funções relacionadas com o sistema de ficheiros em seis grupos:
 - Abertura, criação e fecho de ficheiros
 - Operações sobre ficheiros abertos
 - Operações complexas sobre ficheiros
 - Operações sobre diretórios
 - Operações de gestão do sistema de ficheiros.
 - Acesso a ficheiros mapeados em memória

Sistemas Operativos – DE1 - IST

6

3



O que Ganho/Perco com a Utilização das Funções Sistema

Prós:

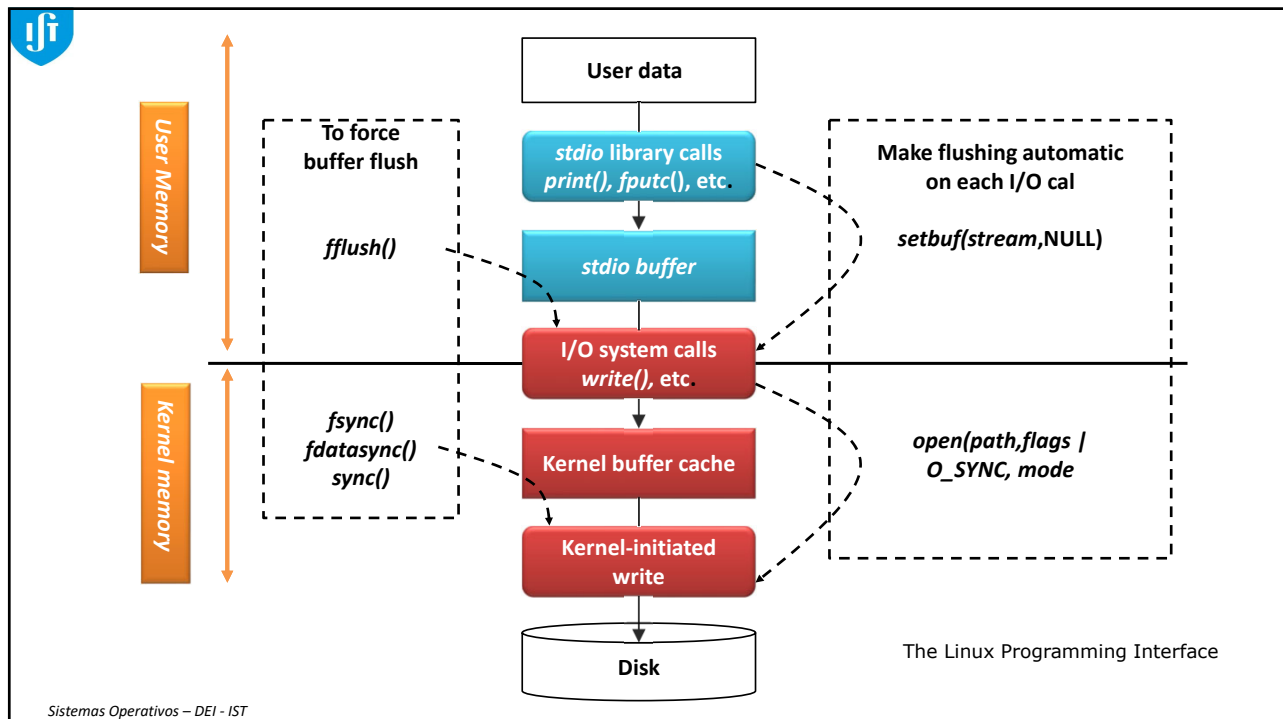
- Em geral, são funções de mais baixo nível, logo permitem maior controlo
- Algumas operações sobre ficheiros só estão disponíveis através desta API

Contras:

- Normalmente, programa que usa `stdio` é mais simples e otimizado

Sistemas Operativos – DE1 - IST

7



Sistemas Operativos – DE1 - IST

8



Abertura, Criação e Fecho de Ficheiros

Operações	Genéricas	Linux	Descrição
Simples	Fd := Abrir (Nome, Modo) Fd := Criar (Nome, Proteção)	<code>int open(const char *path, int flags, mode_t mode)</code>	Abre um ficheiro Cria um novo ficheiro
	Fechar (Fd)	<code>int close(int fd)</code>	Fecha um ficheiro

- Abrir e Criar em Linux é a mesma função
- A função `open()` tem numerosos parâmetros

Sistemas Operativos – DE1 - IST

9



Abrir e Fechar ficheiros


- É mantida uma **Tabela de Ficheiros Abertos** por processo
- Abrir um ficheiro:
 - Pesquisar o diretório
 - Verificar se o processo tem permissões para o modo de acesso que pede
 - Copiar a meta-informação (o *inode*) para memória
 - Devolve ao utilizador um identificador – *file descriptor*, que é usado como referência para essa posição de memória
- Fechar um ficheiro:
 - Liberta a memória que continha a meta-informação do ficheiro
 - Caso necessário, atualiza essa informação no sistema de memória secundária

Processo: instância de um programa em execução

Sistemas Operativos – DE1 - IST

11


5



Privilégios de Acesso aos Ficheiros

Sistemas Operativos – DE1 - IST

13



Autenticação de Processos

- Cada processo corre em nome de um utilizador (UID)/grupo (GID)
- Atribuídos ao primeiro processo criado quando o utilizador se autentica (*log in*)
 - Obtidos do ficheiro `/etc/passwd` no momento do *login*
- Processos filho criados pelos processos desse utilizador herdam UID/GID

Sistemas Operativos – DE1 - IST

14



Controlo dos Direitos de Acesso

- O modelo de autorização mais frequente baseia-se numa Matriz de Direitos de Acesso

Utilizadores	Objetos				Capacidade
	1	2	3	4	
A	ler	-	escrever	executar	
B	-	Ler/escrever	-	-	
C	-	-	-	Ler/escrever	

- Colunas designam-se **Listas de Direitos de Acesso (ACL – Access Control List)**
- Linhas designam-se por **Capacidades**

Sistemas Operativos – DE1 - IST

15



ACL dos Ficheiros em Unix

Nos SGF utiliza-se normalmente listas de controlo de acesso ACL que são guardadas na metadata dos ficheiros

```
$ ls -l
total 7
-rw-r--r-- 1 jpbarreto 197121 113 nov 20 2017 0SXW49A0.cookie
-rw-r--r-- 1 jpbarreto 197121 0 nov 15 2017 container.dat
-rw-r--r-- 1 jpbarreto 197121 91 jan 30 2018 deprecated.cookie
drwxr-xr-x 1 jpbarreto 197121 0 fev 20 2018 DNTException/
drwxr-xr-x 1 jpbarreto 197121 0 jul 15 12:04 ESE/
-rw-r--r-- 1 jpbarreto 197121 119 jan 25 2018 F0RK22C9.cookie
drwxr-xr-x 1 jpbarreto 197121 0 ago 16 23:36 Low/
-rw-r--r-- 1 jpbarreto 197121 95 nov 15 2017 N036J89L.cookie
-rw-r--r-- 1 jpbarreto 197121 470 dez 7 2017 OCP0PE1W.cookie
drwxr-xr-x 1 jpbarreto 197121 0 nov 15 2017 PrivacIE/
-rw-r--r-- 1 jpbarreto 197121 155 jan 26 2018 URPQ8UQ1.cookie
-rw-r--r-- 1 jpbarreto 197121 112 dez 7 2017 ZDCLU5P1.cookie
```

Sistemas Operativos – DE1 - IST

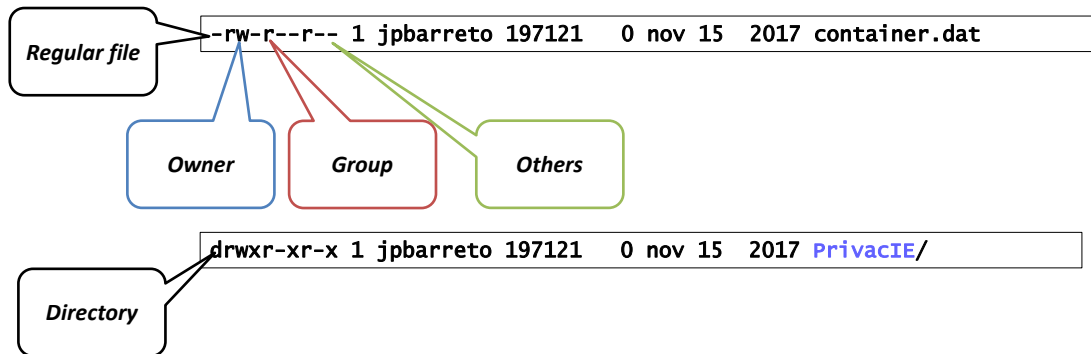
16



ACL em Unix

Simplificação das ACL do Unix considerar três grupos

- Dono
- Grupo
- Restantes utilizadores



Sistemas Operativos – DE1 - IST

17



Controlo dos Direitos de Acesso

1. Processo pede para executar operação sobre objeto gerido pelo núcleo
2. Núcleo valida se na ACL do ficheiro o UID/GID correspondente ao processo tem direitos para executar aquela operação sobre aquele objeto
3. Se sim, núcleo executa operação; se não, retorna erro

Sistemas Operativos – DE1 - IST

19



Copy the file named argv[1] to a new file named in argv[2]

```
#include <sys/stat.h>
#include <fcntl.h>
#include "tapi_hdr.h"

#define BUF_SIZE 1024

int
main(int argc, char *argv[])
{
    int inputFd, outputFd, openFlags;
    mode_t filePerms;
    ssize_t numRead;
    char buf[BUF_SIZE];

    if (argc != 3 || strcmp(argv[1], "--help") == 0)
        usageErr("%s old-file new-file\n", argv[0]);

    /* Open input and output files */

    inputFd = open(argv[1], O_RDONLY);
    if (inputFd == -1)
        errExit("opening file %s", argv[1]);

    openFlags = O_CREAT | O_WRONLY | O_TRUNC;
    filePerms = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |
                S_IROTH | S_IWOTH; /* rw-rw-rw- */
    outputFd = open(argv[2], openFlags, filePerms);
    if (outputFd == -1)
        errExit("opening file %s", argv[2]);
```

Sistemas Operativos – DE1 - IST

20



Operações sobre Ficheiros Abertos

Operações	Genéricas	Linux
Ficheiros Abertos	Ler (Fd, Tampão, Bytes)	int read(int fd, void *buffer, size_t count)
	Escrever (Fd, Tampão, Bytes)	int write(int fd, void *buffer, size_t count)
	Posicionar (Fd, Posição)	int lseek(int fd, off_t offset, int origin)

Sistemas Operativos – DE1 - IST

21



```
int read(int fd, void *buffer, size_t count)
```

- O argumento `count` especifica o número máximo de *bytes* a ler.
- O argumento *buffer* indica o endereço de memória no qual os dados devem ser colocados. Este *buffer* deve ter pelo menos a dimensão indicada em `count`. **As chamadas do sistema não alocam memória para os dados que retornam**
- Uma chamada bem sucedida retorna o número de *bytes* realmente lidos, ou 0 se o fim do ficheiro for encontrado. Em caso de erro o retorno é -1
- Uma chamada a `read()` pode ler menos bytes que o solicitado. No caso de um ficheiro regular, a razão provável é que estávamos perto do fim do ficheiro

Sistemas Operativos – DE1 - IST

22



```
int write(int fd, void *buffer, size_t count)
```

- Os argumentos para `write()` são semelhantes aos de `read()`
 - *buffer* é o endereço dos dados a serem escritos
 - `count` é o número de *bytes* a escrever
 - `fd` é um *filedescriptor* que referencia o ficheiro para o qual os dados devem ser gravados.
- Se tiver sucesso, `write()` retorna o número de bytes realmente escritos, pode ser menos do que `count`.
- Um retorno bem sucedido de `write()` não garante que os dados foram transferidos para o disco, porque o núcleo do SO efetua *buffering* de I/O para o disco, a fim de reduzir as transferências

Sistemas Operativos – DE1 - IST

23



O Cursor

- Para qualquer ficheiro aberto, o núcleo mantém um cursor
- Este é o índice que indica a posição no ficheiro onde a próxima operação de `read()` ou `write()` se executará
- O cursor é expresso como o deslocamento em *bytes* a partir do início (o primeiro *byte* do ficheiro tem o deslocamento 0)
- Avança automaticamente com cada *byte* lido ou escrito

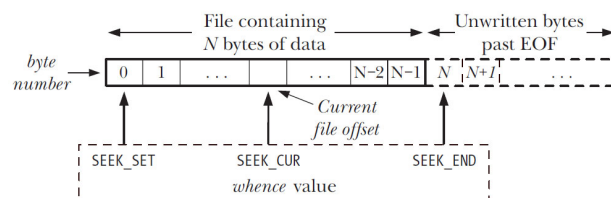
Sistemas Operativos – DEI - IST

24



```
int lseek(int fd, off_t offset, int whence)
```

- O cursor pode ser alterado explicitamente pelo programa
- O `offset` especifica um valor em *bytes*.
-
- O argumento `whence` indica o ponto base a partir do qual o deslocamento deve ser interpretado,



Sistemas Operativos – DEI - IST

25



Copy the file named `argv[1]` to a new file named in `argv[2]`

```
#include <sys/stat.h>
#include <fcntl.h>
#include "tlipi_hdr.h"

#define BUF_SIZE 1024

int
main(int argc, char *argv[])
{
    int inputFd, outputFd, openFlags;
    mode_t filePerms;
    ssize_t numRead;
    char buf[BUF_SIZE];

    if (argc != 3 || strcmp(argv[1], "--help") == 0)
        usageErr("%s old-file new-file\n", argv[0]);

    /* Open input and output files */
    inputFd = open(argv[1], O_RDONLY);
    if (inputFd == -1)
        errExit("opening file %s", argv[1]);

    openFlags = O_CREAT | O_WRONLY | O_TRUNC;
    filePerms = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP |
        S_IROTH | S_IWOTH; /* rw-rw-rw- */
    outputFd = open(argv[2], openFlags, filePerms);
    if (outputFd == -1)
        errExit("opening file %s", argv[2]);
```

```
/* Transfer data until we encounter end of input or
an error */

while ((numRead = read(inputFd, buf, BUF_SIZE))
    > 0)
    if (write(outputFd, buf, numRead) !=
        numRead)
        fatal("couldn't write whole buffer");
    if (numRead == -1) errExit("read");

if (close(inputFd) == -1)
    errExit("close input");
if (close(outputFd) == -1)
    errExit("close output");

exit(EXIT_SUCCESS);
}
```

Sistemas Operativos – DE1 - IST

26



Comando do *Shell* `strace`

- Usar o comando `strace` para ver as chamadas sistema que os programas estão a executar
- Usar para ver a execução deste programa de demonstração

- The `strace` tool provides an awesome way to see what programs are up to. By running it, you can trace which system calls a program makes, see the arguments and return codes, and generally get a very good idea of what is going on.
- The tool also takes some arguments which can be quite useful.
- For example, `-e trace=open,close,read,write` only traces calls to those system calls and ignores all others.
- read the man pages and find out how to harness this wonderful tool.

Sistemas Operativos – DE1 - IST

28



Conclusões

- Este módulo foca a programação de ficheiros com base nas *system calls* disponibilizadas pelo Linux
- A *system call* `open ()` é muito relevante pela elevado grau de parametrização e pela sua relação com o modelo de permissões do Linux que se aplica aos ficheiros e à generalidade dos objetos do núcleo (ACL aplicadas a dono, grupo e outros)
- As restantes *system calls* são muito semelhantes às que conheciam da utilização da `stdio`. Mas necessitam de alguma atenção para perceber os detalhes da semântica de execução e o significado dos parâmetros

Sistemas Operativos – DE1 - IST