Sistemas de Ficheiros

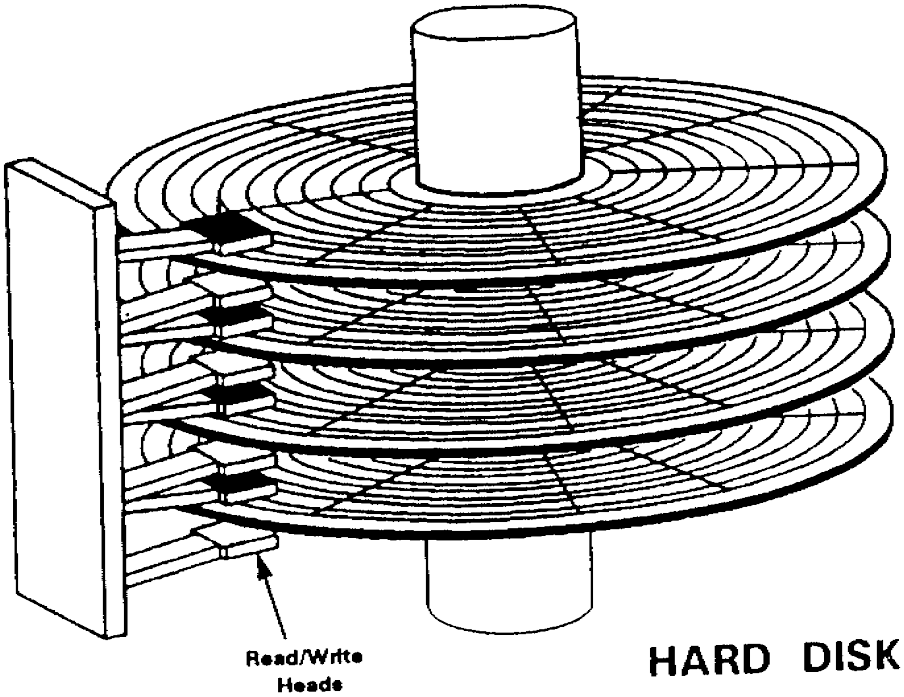
Sistemas Operativos

2021 - 2022

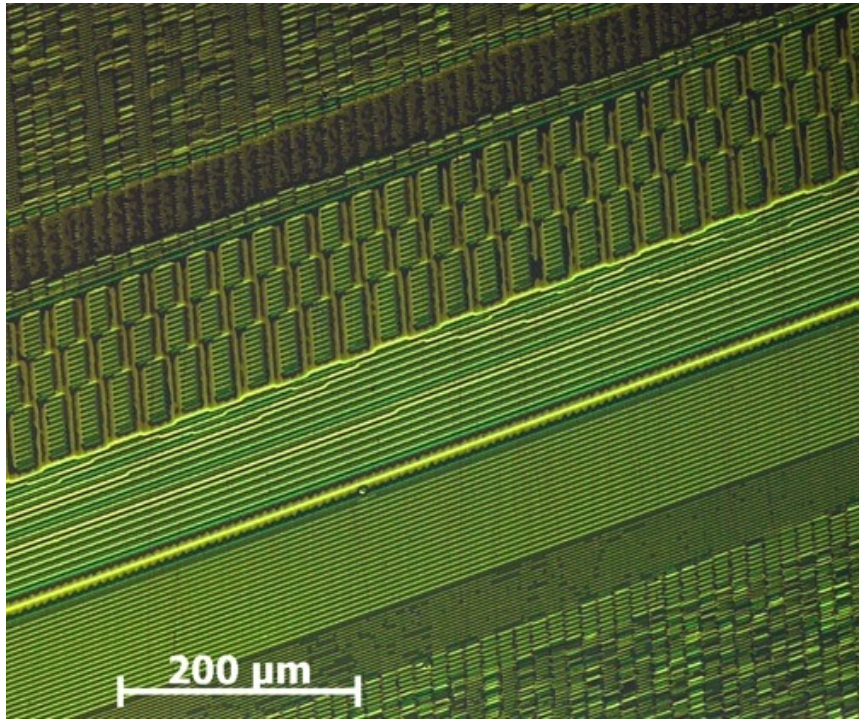
A abstração: Ficheiro

- Colecção de dados persistentes, geralmente relacionados, identificados por um nome
- Organizado em hierarquia de pastas

A realidade



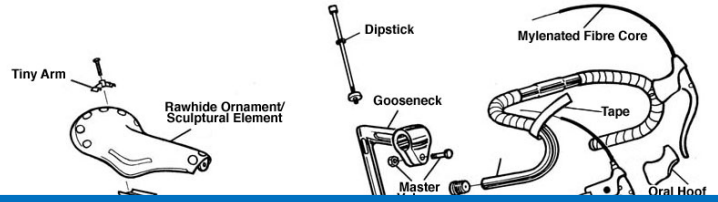
HARD DISK



Plano das próximas aulas

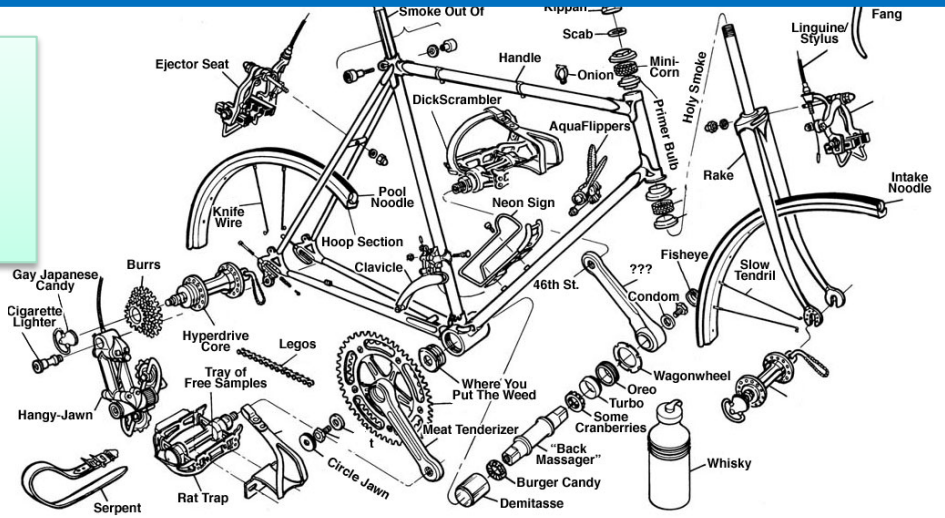
Aprender a usar os sistemas de ficheiros (abstrações, APIs)

1ª Parte



Introduzir a organização interna dos sistemas de ficheiros:
- **relevante para o projeto**

2ª Parte

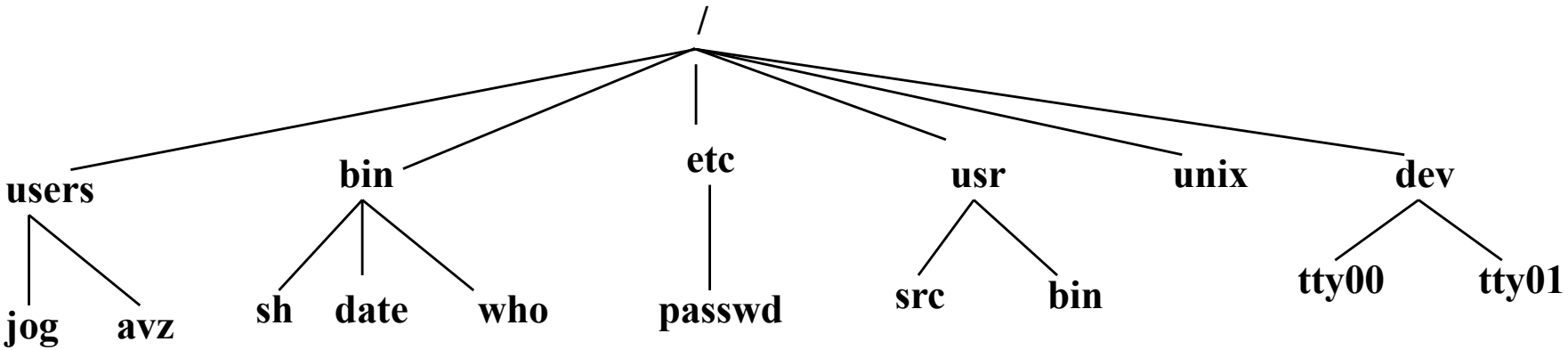


Sistema de Ficheiros

- Composto por um conjunto de entidades fundamentais:
  - um sistema de organização de nomes para identificação dos ficheiros;
  - uma interface programática para comunicação entre os processos;
  - sistema de ficheiros

Árvore de diretórios

- Mantém a meta-informação sobre ficheiros
  - no mesmo sistema de memória secundária que a informação que descreve
  - entre outros, estabelece a associação entre o nome e um identificador numérico do ficheiro

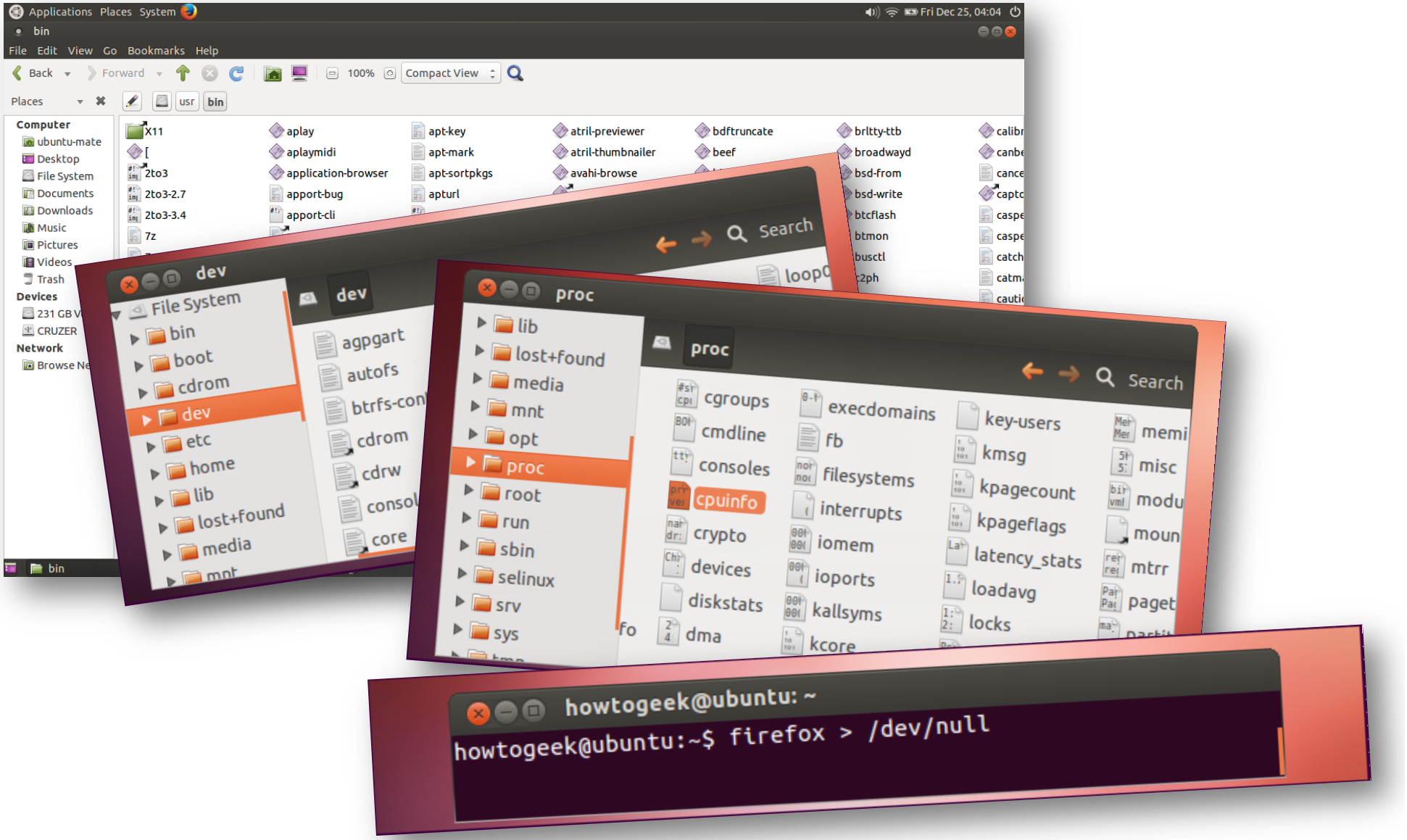


O que é um ficheiro?

```
ls

-rwxr-xr-x  1 luis  staff  8680 Nov 14 19:46 do_exec
```

“Everything is a file”



O princípio *"Everything is a file"*

- Objetos que o SO gere são acessíveis aos processos através de *descritores de ficheiro*
  - Ficheiros, diretorias, dispositivos lógicos, canais de comunicação, etc.
- Vantagens para os utilizadores/programadores
  - Modelo de programação comum
  - Modelo de segurança comum
- Um dos princípios chave do Unix
  - Seguido por muitos SOs modernos
  - Algumas excepções (até no Unix)

Nomes absolutos e nomes relativos

- Nomes absolutos:
  - caminho de acesso desde a raiz
  - Exemplo:

`/home/joao/SO/project.zip`

Mas ter de fornecer sempre o nome absoluto de um ficheiro é fastidioso e pouco flexível...

- Nomes relativos:
  - caminho de acesso a partir do diretório corrente
  - diretório corrente mantido para cada processo como parte do seu contexto
  - Exemplos:

`./SO/project.zip`
(supondo que o diretório corrente é **`/home/joao`**)

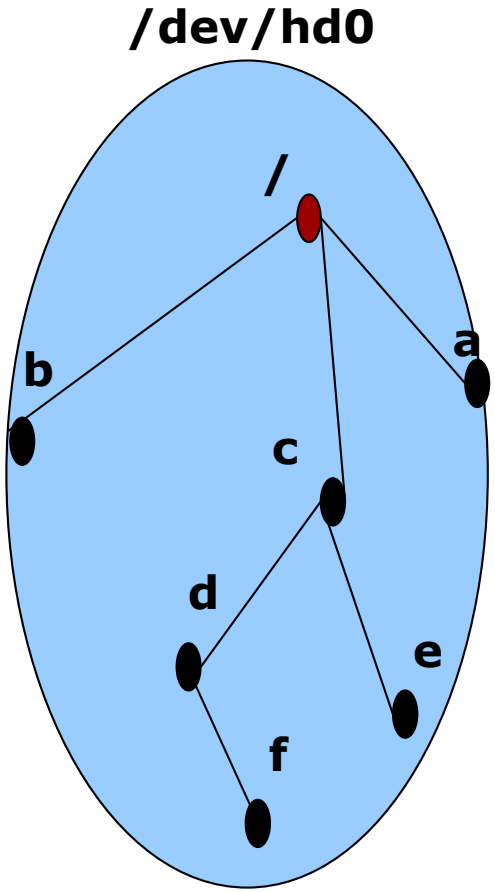
`../project.zip`
(supondo que o dir. corrente seja **`/home/joao/teo`**)

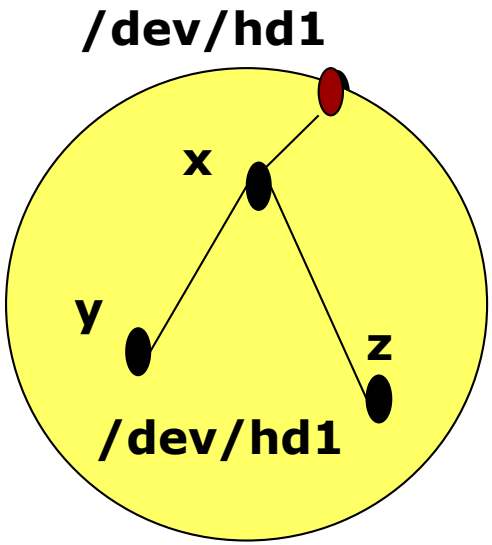
Nomes vs. ficheiros

- Um ficheiro pode ser conhecido por vários nomes:
  - é possível designar o mesmo ficheiro com o nome /a/b/c e com o nome /x/y.
  - é comum chamar a cada um destes nomes *links*
- Problema:
  - quando se pretende apagar o ficheiro com o nome /a/b/c.
  - apagar o conteúdo do ficheiro ou apenas o nome?

- A semântica utilizada na maioria dos sistemas de ficheiros é a última

Como organizar múltiplos sistemas de ficheiros?

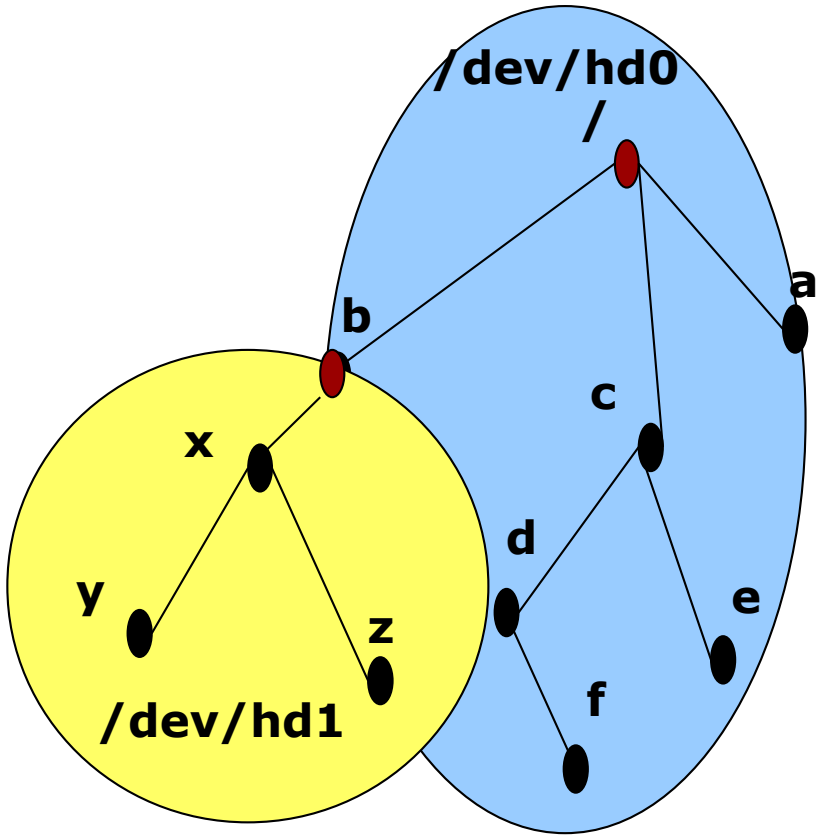





```
mount -t <filesystem> /dev/hd1 /bv
```

Como organizar múltiplos sistemas de ficheiros?

```
mount -t <filesystem> /dev/hd1 /b
```



- *Mount*:
  - liga a raiz do novo sistema de ficheiros a um directório do sistema de ficheiros base

Atributos de um Ficheiro

- Para além do tipo, a meta-informação do ficheiro possui usualmente os seguintes atributos:
  - Protecção
    - quem pode aceder ao ficheiro e quais as operações que pode realizar.
  - Identificação do dono do ficheiro
    - geralmente quem o criou.
  - Dimensão do ficheiro
  - Data de criação, última leitura e última escrita

Programar com ficheiros

Como manipular ficheiros?

- As operações mais frequentes sobre ficheiros são a leitura e escrita da sua informação
- Bastam estas funções para ler e escrever?

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
  - Copia a meta-informação para memória (incluindo o modo de acesso solicitado)
  - Devolve ao utilizador um identificador que é usado como referência para essa posição de memória
- Ler e escrever sobre ficheiros abertos:
  - Dado o identificador de ficheiro aberto, permite obter rapidamente o descritor do ficheiro em memória
- Fechar do 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

Primitivas do Sistema de Ficheiros

- 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 directórios;
  - Acesso a ficheiros mapeados em memória;
  - Operações de gestão dos sistemas de ficheiros.

Abertura, criação e fecho de ficheiros

Retorno	Nome	Parâmetros	Descrição
fd :=	Abrir	(Nome, Modo)	Abre um ficheiro
fd :=	Criar	(Nome, Proteção)	Cria um novo ficheiro
	Fechar	(Fd)	Fecha um ficheiro

Operações sobre ficheiros abertos

Nome	Parâmetros	Descrição
Ler	(Fd, buffer, bytes)	Lê de um ficheiro para um buffer de memória
Escrever	(Fd, buffer, bytes)	Escreve um buffer para um ficheiro
Posicionar	(Fd, Posição)	Posiciona o cursor de leitura ou escrita

Operações complexas sobre ficheiros

- Algumas operações sobre ficheiros permitem realizar operações sobre a totalidade do ficheiro, como copiá-lo, apagá-lo ou movê-lo

Nome	Parâmetros	Descrição
Copiar	(Origem, Destino)	Copia um ficheiro
Mover	(Origem, Destino)	Move um ficheiro de um directório para outro
Apagar	(Nome)	Apaga um ficheiro
LerAtributos	(Nome, Tampão)	Lê atributos de um ficheiro
EscreverAtributos	(Nome, Atributos)	Modifica os atributos

Operações sobre directórios

Nome	Parâmetros	Descrição
ListaDir	(Nome, Tampão)	Lê o conteúdo de um directório
MudaDir	(Nome)	Muda o directório por omissão
CriaDir	(Nome, Protecção)	Cria um novo directório

Os canais standard

- Inicialmente, tabela de ficheiros de um processo preenchida com 3 ficheiros abertos:
  - stdin, stdout, stderr
- Normalmente, referenciam os canais de *input* e *output* da consola em que o processo foi lançado
- ...Mas nem sempre!

```
foo < out.txt
ls > listagem.txt
foo >& erros.txt
```

Modelo de programação:
A API do sistema de ficheiros
(Revisão de IAED)

Abordagem 1:
Trabalhar com ficheiros usando as funções da stdio

Operações sobre ficheiros

- Até este momento fizemos sempre leituras do stdin e escrevemos sempre para o stdout. Vamos ver agora como realizar estas operações sobre ficheiros.
- Como abrir um ficheiro?

```
FILE *fp;

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

Ponteiro para estrutura que representa o ficheiro aberto

Modo de abertura do ficheiro. Neste caso estamos a abrir o ficheiro em modo de leitura

Operações sobre ficheiros

- Até este momento fizemos sempre leituras do stdin e escrevemos sempre para o stdout. Vamos ver agora como realizar estas operações sobre ficheiros.
- Como abrir um ficheiro?

Modos de abertura

r – abre para leitura (read)

w – abre um ficheiro vazio para escrita (o ficheiro não precisa de existir)

a – abre para acrescentar no fim (“append” ; ficheiro não precisa de existir)

r+ – abre para escrita e leitura; começa no início; o ficheiro tem de existir

w+ – abre para escrita e leitura (tal como o “w” ignora qualquer ficheiro que exista com o mesmo nome, criando um novo ficheiro)

a+ – abre para escrita e leitura (output é sempre colocado no fim)

…mas há mais

Exemplo

```
#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;
}
```

Se não conseguir abrir, fp fica igual a NULL

Exemplo

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

int main()
{
  FILE *fp;

  fp = fopen("teste.txt", "r");
  if (fp == NULL) {
    perror("teste.txt");
    exit(1);
  }

  return 0;
}
```

Escreve a mesma mensagem de erro.

perror() escreve no "standard error" (stderr) a descrição do último erro encontrado na chamada a um sistema ou biblioteca.

Exemplo 2

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

int main()
{
  FILE *fp;

  fp = fopen("teste.txt", "r");
  if (fp == NULL) {
    perror("teste.txt");
    exit(1);
  }

  fclose(fp);

  return 0;
}
```

Fecha o ficheiro

Exemplo 3

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

int main()
{
  FILE *fp;

  fp = fopen("teste.txt", "w");
  if (fp == NULL) {
    perror("teste.txt");
    exit(1);
  }

  fprintf(fp, "Hi file!\n");

  fclose(fp);

  return 0;
}
```

Escreve para um ficheiro

Exemplo 3

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

int main()
{
  FILE *fp;

  fp = fopen("teste.txt", "w");
  if (fp == NULL) {
    perror("teste.txt");
    exit(1);
  }

  fputs("Hi file!", fp);

  fclose(fp);

  return 0;
}
```

Escreve para um ficheiro (alternativa)

Exemplo 4

```
#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 floats guardados num ficheiro

Exemplo 5

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

int main()
{
  FILE *myfile; int i;
  myfile = fopen("info.dat", "a");

  for (i=0;i<100;i++)
     fprintf(myfile,"%d\n",i);

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

Adiciona um conjunto de 100 inteiros ao fim de um ficheiro

O cursor

- Para qualquer ficheiro aberto, é mantido um cursor
  - Avança automaticamente com cada byte lido ou escrito
- Para sabermos em que posição estamos, usar função ***ftell***

```
long ftell(FILE *stream);
```

- Para repor o cursor noutra posição, usar função ***fseek***
  - Por exemplo, colocar cursor no início ou final do ficheiro

```
int fseek(FILE *stream, long offset, int whence);
```

Escritas são imediatamente persistentes?

- Após escrita em ficheiro, essa escrita está garantidamente persistente no disco?
  - Nem sempre!
  - Para optimizar o desempenho, escritas são propagadas para disco tardiamente
- Função ***fflush*** permite ao programa forçar que escritas feitas até agora sejam persistidas em disco
  - Função só retorna quando houver essa garantia
  - Função demorada, usar apenas quando necessário

```
int fflush(FILE *stream);
```

Modelo de programação:
A API do sistema de ficheiros
(Revisão de IAED)

Abordagem 2:
Trabalhar com ficheiros usando as funções da API do sistema de ficheiros do Unix

O que ganho/perco?

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 optimizado
  - Discutiremos mais à frente em SO porque é que stdio é mais optimizado

Sistema de Ficheiros do Unix

Operações	Genéricas	Linux
Simples	Fd := Abrir (Nome, Modo) Fd := Criar (Nome, Protecção)	int open(const char *path, int flags, mode_t mode)
	Fechar (Fd)	int close(int fd)
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)
Complexas	Criar link (Origem, Destino)	int symlink(const char *oldpath, const char *newpath) int link(const char *oldpath, const char *newpath)
	Mover (Origem, Destino)	int rename(const char *oldpath, const char *newpath)
	Apagar link (Nome)	int unlink(const char *path)
		int dup(int fd), int dup2(int oldfd,int newfd)
	LerAtributos (Nome, Tampão)	int stat(const char *path, struct stat *buffer)
	EscreverAtributos (Nome, Atributos)	int fcntl(int fd,int cmd,struct flock *buffer) int chown(const char *path, uid_t uid, gid_t gid) int chmod(const char *path, mode_t mode)
Ficheiros em memória	MapearFicheiro(Fd,pos,endereço,dim)	void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset)
	DesMapearFicheiro(endereço,dim)	int munmap(void *addr, size_t len)
Directórios	ListaDir (Nome, Tampão)	int readdir(int fd, struct dirent *buffer, unsigned int count)
	MudaDir (Nome)	int chdir(const char *path)
	CriaDir (Nome, Protecção)	int mkdir(const char *path, mode_t mode)
	RemoveDir(Nome)	int rmdir(const char *path)
Sistemas de Ficheiros	Montar (Directório, Dispositivo)	int mount(const char *device, const char *path, const char *fstype, unsigned long flags, const void *data)
	Desmontar (Directório)	int umount(const char *path)