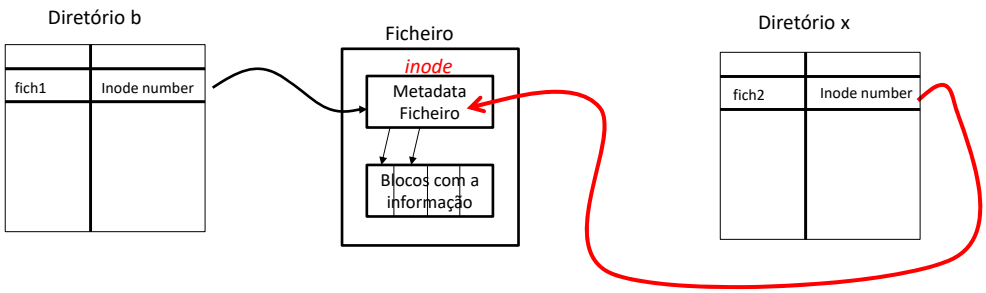
Programação Sistema de Ficheiros

Diretórios e Funções Avançadas

Diretórios e Nomes de Ficheiros

- Um ficheiro pode ser conhecido por vários nomes:
  - Pode-se designar o mesmo ficheiro com o nome `/a/b/fich1` e com o nome `/x/fich2`.
  - É comum chamar a cada um destes nomes *links*

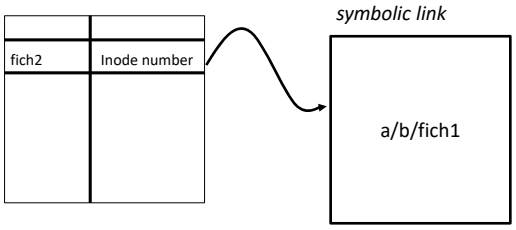


Links: *hard Link*

- Entradas em diversos diretórios apontam para o mesmo ficheiro (o mesmo *inode*) pelo que, do ponto de vista do sistema, são indistinguíveis
- Corresponde ao conceito de cópia de um ficheiro (sem cópia real dos dados)
- Se apagarmos um ficheiro com vários *hard links*, o ficheiro continua a existir. Só será removido quando o ultimo *hard link* for apagado
- A chamada sistema `unlink()` usada para eliminar ficheiros, tem este nome porque precisamente o que faz é eliminar um *link* e não forçosamente apagar o ficheiro

Symbolic Link

- O *symbolic link* é um ficheiro (de tipo diferente) que contem o caminho de acesso para o ficheiro original
- Se o ficheiro original for apagado o *link* fica quebrado
- Mais genérico na utilização, podendo ser usado para diretórios ou entre partições enquanto os hard link estão limitados a uma partição



Indica um ficheiro tipo *symbolic link*

```
ubuntu@ubuntu:~/tlpi-190930-dist/tlpi-dist/teste$ ls -al
total 12
drwxrwxr-x  2 ubuntu ubuntu 4096 Oct 21 17:18 .
drwxrwxr-x 51 ubuntu ubuntu 4096 Oct 21 17:17 ..
-rw-rw-r--  1 ubuntu ubuntu   17 Oct 21 17:18 Exemplo1
lrwxrwxrwx  1 ubuntu ubuntu    8 Oct 21 17:18 Exemplo2 -> Exemplo1
```

Funções de Gestão dos Ficheiros

- Percebendo o conceito de *hard* e *symbolic link* torna-se simples entender a funcionalidade das seguintes chamadas sistema

Operações	Linux
Copiar (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 (Nome)	`int unlink(const char *path)`

Funções sobre os diretórios

	Operações	Linux
Diretórios	ListaDir (Nome , Buffer)	int readdir (int fd, struct dirent *buffer, 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)

Descritores Individuais de Ficheiros (*i-nodes*)

- Um ficheiro é univocamente identificado, dentro de cada partição, pelo número de *i-node*
- Os directórios só têm que efetuar a ligação entre um nome do ficheiro e o número do seu descritor

	Número do Inode	Dimensão do Registo	Dimensão do nome	Tipo	Nome							
0	54	12	1	2	.	\0	\0	\0				
12	79	12	2	2	.	.	\0	\0				
24	23	16	6	1	c	a	r	l	o	s	\0	\0
40	256	16	7	1	m	a	r	q	u	e	s	\0

Programação com Diretórios

- Os diretórios são lidos sequencialmente passando de um entrada para a seguinte
- Os diretórios podem ter um tratamento programático dos diretórios
- Para abrir um ficheiro diretório usa-se a função `opendir()` que retorna uma estrutura de dados que facilita a leitura, designada DIR é um *handle* para um *directory stream* que é passado as funções que leem o diretório

```
#include <dirent.h>
DIR *opendir(const char *dirpath);
```

Leitura dos Diretórios

```
struct dirent *readdir(DIR *dirp);
```

- Cada chamada a `readdir()` lê a próxima entrada do diretório referenciado por `dirp` e devolve um ponteiro para uma estrutura de tipo `dirent`, que contem as seguintes informações

```
struct dirent {
ino_t d_ino;     /* File i-node number */
char d_name[];   /* Null-terminated name of file */
};
```

- No fim do diretório ou erro, `readdir()` retorna `NULL`, neste último caso, `errno` indica o erro.

struct stat

```
struct stat {
dev_t     st_dev;   /*        IDs of device on which file resides */
ino_t     st_ino;   /*        I-node number of file */
mode_t    st_mode;  /*        File type and permissions */
nlink_t   st_nlink; /*        Number of (hard) links to file */
uid_t     st_uid;   /*        User ID of file owner */
gid_t     st_gid;   /*        Group ID of file owner */
dev_t     st_rdev;  /*        IDs for device special files */
off_t     st_size;  /*        Total file size (bytes) */
blksize_t st_blksize;         /*        Optimal block size for I/O (bytes) */
blkcnt_t  st_blocks;          /*        Number of (512B) blocks allocated */
time_t    st_atime; /*        Time of last file access */
time_t    st_mtime; /*        Time of last file modification */
time_t    st_ctime; /*        Time of last status change */
};
```

Esta estrutura de dados tem todas as informações em formato binário se as pretendermos mostrar a utilizadores é necessário fazer *parsing*

Listar um diretório

```
DIR *dirp;
struct dirent *dp;
dirp = opendir(dirpath);
if (dirp == NULL) {
        errMsg("opendir failed on '%s'", dirpath);
        return;
        }
for (;;) {
        errno = 0; /* To distinguish error from end-of-directory */
        dp = readdir(dirp);
        if (dp == NULL)
                break;
        if (strcmp(dp->d_name, ".") == 0 || strcmp(dp->d_name, "..") == 0)
                continue; /* Skip . and .. */
        printf("%s\n", dp->d_name);
}
```


Ler e Modificar Atributos

Comando para ver os atributos de um ficheiro

- O comando `stat` permite ver a informação de um ficheiro

```
ubuntu@ubuntu:~/Fichteste$ echo Exemplo para ver Stat > exemplostat
ubuntu@ubuntu:~/Fichteste$ stat exemplostat
  File: exemplostat
  Size: 22              Blocks: 8          IO Block: 4096   regular file
Device: 821h/2081d      Inode: 516217      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/  ubuntu)   Gid: ( 1001/  ubuntu)
Access: 2022-10-25 09:57:42.345926191 +0000
Modify: 2022-10-25 09:57:42.345926191 +0000
Change: 2022-10-25 09:57:42.345926191 +0000
 Birth: -
```


Estado do Ficheiro: *system calls*

- A mesma informação pode ser obtida programaticamente através de um conjunto de *system calls* que preenchem um *buffer* passado como parâmetro da função

- Para executá-las é necessário ter direitos de acesso à diretória (não é necessário ao ficheiro)

```
int stat(const char *restrict pathname, struct stat *restrict statbuf);

int fstat(int fd, struct stat *statbuf);

int lstat(const char *restrict pathname, struct stat *restrict statbuf);
```

Mudança de Permissões

- O comando `chmod` permite mudar as permissões de acesso ao ficheiro

```
ubuntu@ubuntu:~/Fichteste$ ls -l exemplostat
-rw-rw-r-- 1 ubuntu ubuntu 22 Oct 25 09:57 exemplostat
ubuntu@ubuntu:~/Fichteste$ chmod u-rw exemplostat
ubuntu@ubuntu:~/Fichteste$ ls -l exemplostat
----rw-r-- 1 ubuntu ubuntu 22 Oct 25 09:57 exemplostat
```

- As mesmas operações podem ser executadas programaticamente com a função `chmod()`

```
#include <sys/stat.h>
int chmod(const char *pathname, mode_t mode)
```

- A mudança de permissões e a sua verificação tem numerosos detalhes, importantes por questões de segurança

Operações Globais sobre o Sistema de Ficheiros

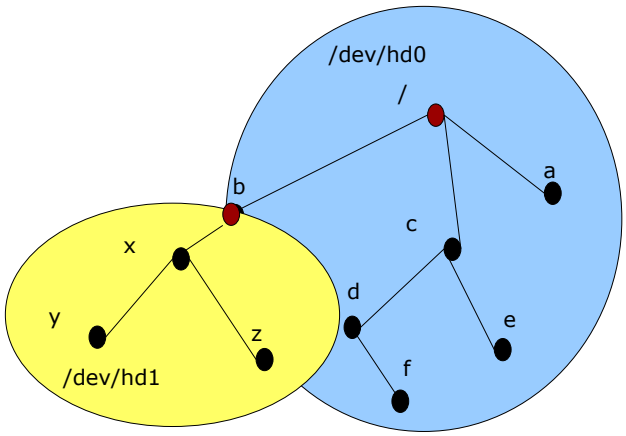
Montar um volume na hierarquia de diretórios

Comando Mount

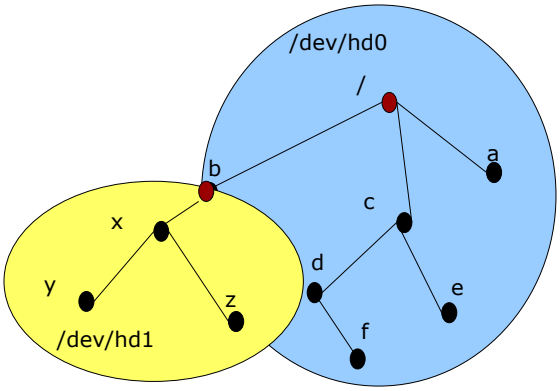
- A maioria dos periféricos detetados pelo sistema operativo possui um nome no diretório especial `/dev`
- Os dispositivos de memória secundária são periféricos tal como as impressoras, o teclado ou o ecrã, pelo que também possuem um nome neste diretório.
- Para aceder aos sistemas de ficheiros dos dispositivos, é necessário montá-lo no sistema de ficheiros primário. Esta operação consiste em ligar a raiz do novo sistema de ficheiro a um diretório do sistema de ficheiros raiz.
- Quer o dispositivo de base da operação de montagem quer o dispositivo que é montado possuem uma árvore de diretórios com uma raiz única.

Como Organizar Múltiplos Sistemas de Ficheiros?

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



Como Organizar Múltiplos Sistemas de Ficheiros?



- Após a operação de montagem o diretório b do dispositivo `/dev/hd0` e a raiz do dispositivo `/dev/hd1` passam a ser o mesmo diretório
- Os ficheiros no dispositivo `/dev/hd1` ficam posteriormente acessíveis através dessa raiz,

`mount`

- A beleza conceptual do `mount` é que em vez de ter um número de sistemas de ficheiros separados, `mount` unifica todos os sistemas numa árvore, tornando a nomenclatura uniforme e conveniente
- Para ver o que está montado no sistema e em que pontos, basta executar o comando `mount`

```
/dev/sda1 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
/dev/sda5 on /tmp type ext3 (rw)
/dev/sda7 on /var/vice/cache type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
AFS on /afs type afs (rw)
```

- Como é natural existe a chamada sistema `mount()` para efetuar esta operações programaticamente

Conclusões

- A relação entre os ficheiros e os nomes nos diretórios é estabelecida através de *links.* Existem dois tipos de links, *hard link* e *symbolic link* que são conceitos diferentes.
- Os diretórios são ficheiros com uma estrutura própria, podem ser lidos e pesquisados programaticamente.
- O estado dos ficheiros pode ser consultado e alterado em particular a ACL que controla a sua proteção.
- Existem operações globais sobre o SFG em particular o *mount* que permite montar numa hierarquia única diferentes sistemas de ficheiros.