

Accesso al file system

- Il file system è una struttura gerarchica ‘ad albero/grafico’ composta da directory e file. Il kernel, tramite la libreria C, mette a disposizione molte funzioni per la *navigazione* nel file system e per l’accesso ai file.
- Le funzioni di navigazione comprendono lo spostamento nelle directory, l’accesso all’elenco di file contenuti in una directory e la creazione di nuove directory.
- Le funzioni di accesso ai file permettono di aprire, creare, cancellare, scrivere e leggere file secondo due ‘filosofie’ complementari:
 - mediante l’astrazione *stream*
 - mediante funzioni di basso livello

Accesso al contenuto di una directory

- L'accesso ad una directory richiede una fase di *apertura*, nella quale si associa il percorso della directory ad un handler di tipo DIR*
 - DIR *dp = opendir(char *pathname)
- una iterazione fra i vari membri della directory acceduti mediante la struttura dirent, ad esempio in un ciclo while
 - struct dirent *de = readdir(DIR *dp)
- una fase di chiusura
 - void closedir (DIR *dp)
- L'accesso ai membri mediante la funzione readdir è sequenziale (ogni invocazione restituisce il dirent successivo) e termina con un valore di ritorno nullo.

La struttura dirent

- Ogni elemento ritornato dall'invocazione di `readdir` è di tipo *struct dirent* e contiene le informazioni del *nodo* interno alla directory.
- Lo standard POSIX raccomanda almeno la presenza del campo *char *d_name* che, ovviamente, rappresenta il nome dell'elemento.
- L'implementazione specifica della libreria C può fornire anche informazioni aggiuntive. Si noti che l'utilizzo di queste informazioni può ostacolare la portabilità dell'applicazione.
- Dal valore di *d_name* è possibile ottenere tutte le caratteristiche del nodo (tipo, dimensione, ecc...) mediante la funzione:
 - `int stat(const char *filename, struct stat *buf)`
- **Attenzione:** *buf* deve essere allocato dal chiamante.

aspetto di dirent e stat

```
/* Lay out della struttura dirent ritornata da 'readdir' */
struct dirent {
    long          d_ino; /* inode number */
    off_t         d_off; /* offset to the next dirent */
    unsigned short d_reclen; /* length of this record */
    unsigned char  d_type; /* type of file */
    char          d_name[256]; /* filename */
};
/* Lay out della struttura stat ritornata da 'stat' */
struct stat {
    dev_t         st_dev; /* device */
    ino_t         st_ino; /* inode */
    mode_t        st_mode; /* protection and type */
    nlink_t       st_nlink; /* number of hard links */
    uid_t         st_uid; /* user ID of owner */
    gid_t         st_gid; /* group ID of owner */
    dev_t         st_rdev; /* device type (if inode device) */
    off_t         st_size; /* total size, in bytes */
    unsigned long st_blksize; /* blocksize for filesystem I/O */
    unsigned long st_blocks; /* number of blocks allocated */
    time_t        st_atime; /* time of last access */
    time_t        st_mtime; /* time of last modification */
    time_t        st_ctime; /* time of last change */
};
```

clone di ls

```
/* **** */
* $Id: es25.c,v 1.1 2002/02/15 17:07:19 valealex Exp $
* Semplice clone di 'ls'
* Compilare con gcc -o es25 es25.c
* **** */
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>
int main(void);
int main() {
    DIR *dp; /* directory ref. */
    struct dirent *de; /* entry ref. */
    if((dp = opendir("."))!=NULL) {
        printf("Error on opening dir\n"); exit(-1);
    }
    while((de=readdir(dp))!=NULL) {
        if(de->d_name[0]!='.') {
            printf("%s\n",de->d_name);
        }
    }
    closedir(dp);
}
```

makefile per es25 (ls)

```
# #####  
# Makefile per la compilazione in due fasi di es25  
# $Id: es26,v 1.1 2002/02/15 17:08:02 valealex Exp $  
# #####  
  
all: es25  
  
es25.o: es25.c  
    gcc -c -o $@ $<  
  
es25: es25.o  
    gcc -o $@ $<  
  
clean:  
    if [ -f es25.o ]; then rm es25.o; fi  
    if [ -f es25 ]; then rm es25; fi
```

Modifica di directory

- Creazione di una directory: `mkdir()`
- Rimozione di una directory (vuota): `rmdir()`
- Link di un nuovo nome ad un file: `link()`
- Cancellazione di file: `unlink()` o `remove()`
 - nota: `unlink` è il nome consigliato dallo standard POSIX
 - `remove` è il nome consigliato dallo standard ANSI
 - ... gli standard sono una opinione ?!
- Cambio nome: `rename()`

rinomina con link/unlink

```
/* **** */
* $Id: es27.c,v 1.1 2002/02/15 17:07:19 valealex Exp $
* Semplice clone di 'mv'
* Compilare con gcc -o es27 es27.c
* **** */
#include <unistd.h>
int main(int argc, char **argv);

int main(int argc, char **argv) {
    if(0==link(argv[1],argv[2])) {
        unlink(argv[1]);
    }
}
```

Stream

- L'accesso ai file mediante l'astrazione stream è preferibile quando il file è utilizzato come *sequenza di byte*.
- L'accesso si ottiene procurandosi un descrittore di tipo FILE*:
 - invocando esplicitamente la funzione `fopen(filename)`
 - utilizzando uno degli *standard stream* aperti come preambolo dell'invocazione del `main()`, ovvero **stdin**, **stdout** e **stderr**.
- I descrittori `stdin`, `stdout` e `stderr` rappresentano i canali predefiniti per l'input, l'output ed i messaggi di errore dell'applicazione e, normalmente, coincidono con la console (video e tastiera).
- Come noto, è possibile richiedere alla shell la *ridirezione* di questi canali su file o pipe che risulterà trasparente all'applicazione (fatta eccezione per l'I/O su terminale).

Accesso ad uno stream

- Uno stream può essere utilizzato per prelevare o depositare caratteri.
- Le funzioni elementari di I/O su stream sono:
 - `fputc(int ch, FILE *stream)`, invia il carattere 'ch' allo stream
 - `int ch=fgetc(FILE *stream)`, preleva un carattere dallo stream
- l'uso del tipo `int` permette la gestione di condizioni particolari (es. fine del file).
- Un accesso molto comodo agli stream si ottiene per mezzo delle funzioni di input ed output *formattato* ovvero:
 - `fprintf`
 - `fscanf`
- La chiusura di uno stream si ottiene con `fclose()`

Accesso a file con funzioni di basso livello

- E' possibile accedere ai file con funzioni di basso livello che consentono un maggiore controllo dell'I/O a discapito della comodità di uso.
- Le funzioni di accesso sono mappate direttamente nelle corrispondenti entry del device driver associato al file.
- L'accesso si ottiene procurandosi un handle al file (in questo caso non più un FILE* ma un intero) in una delle seguenti modalità:
 - invocando esplicitamente la funzione `open(filename)`
 - utilizzando uno degli *standard file handler* aperti come preambolo dell'invocazione del `main()`, ovvero **0,1 e 2**.
- Vi è corrispondenza fra 0 e `stdin`, 1 e `stdout`, 2 e `stderr`

La funzione 'read'

- La lettura di caratteri da un file di cui si conosce l'handler si ottiene con la funzione read:

```
ssize_t read (int filedes, void *buffer, size_t  
bufsize);
```

- legge un massimo di bufsize caratteri dal file indicato da filedes, li deposita nel buffer e restituisce il numero di caratteri effettivamente letti.
- Il valore di ritorno è nullo se non è possibile prelevare caratteri, ad esempio se si è giunti alla fine del file
- Il valore di ritorno è negativo in caso di errore; è possibile identificare la causa dell'errore accedendo alla variabile *errno*

La funzione 'write'

- La scrittura di caratteri su un file di cui si conosce l'handler si ottiene con la funzione write:

```
ssize_t write (int filedes, void *buffer, size_t  
bufsize);
```

- scrive un massimo di bufsize caratteri nel file indicato da filedes, prelevandoli da buffer e restituisce il numero di caratteri effettivamente scritti
- un valore di ritorno non negativo ma inferiore a bufsize indica una possibile mancanza di spazio nel file
- un valore di ritorno negativo (-1) indica una condizione di errore che può essere identificata accedendo alla variabile *errno*

buffering

- L'astrazione stream contempla la *bufferibilità* delle operazioni di lettura e scrittura su file eseguite con le primitive stream. Questa *bufferizzazione* è totalmente indipendente da quella operata eventualmente dal device driver, ad esempio per impacchettare correttamente i frame di trasmissione, e viene eseguita a livello di libreria.
- Tipicamente, ad esempio, la libreria *disturba* il driver che controlla le operazioni di scrittura a video solo quando incontra un fine linea (CR) e rimanda la scrittura effettiva dei caratteri precedenti. E' possibile comunque forzare la scrittura immediata di uno stream con la funzione `fflush()`.
- La conoscenza di questo diverso comportamento è fondamentale in applicazioni *real time*.

effetto del buffering

```
/* Test for buffering system in streams vs low level files */
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

int main(void);

int main() {
    char buffer[1];

    printf("s");
    sleep(3);
    printf("\n");
    buffer[0] = 'l';
    write(1,buffer,1);
    sleep(3);
    buffer[0] = '\n';
    write(1,buffer,1);
    printf("All done\n");
    return(0);
}
```

Posizionamento e mixing

- Le operazioni di lettura e scrittura su file ad accesso *random* fanno uso di un *indice* che si può spostare con le funzioni `lseek` (basso livello) e `fseek` (stream).
- Ovviamente non ha senso parlare di indice per file strettamente sequenziali come l'input da tastiera o da interfaccia seriale.
- E' possibile, anche se sconsigliabile, mescolare gli accessi ad un file fra stream e low-level.
- Da uno stream è possibile ottenere l'handle con la funzione *fileno*, mentre il contrario si ottiene con la funzione *fdopen*.