ALCUNE CHIAMATE DI SISTEMA PER LA GESTIONE DEL FILE SYSTEM

- CREAZIONE

int creat(char *path, int perms)

ritorna fd o –1

Pseudocodice dell'operazione:

```
If(file esiste)
        If(Write permission)
                Tronca la lunghezza a 0 e lascia il file aperto in scrittura;
        Else
                Exit(-1);
Else
        Alloca un nuovo INODE con link=1 e permessi in scrittura
```

- CANCELLAZIONE

int unlink(char *path)

Decrementa il Link-count dell'Inode. Ritorna 0 oppure –1. Pseudocodice:

```
If (link-count == 0)    && (nessun altro processo ha il file aperto)
                        && (il fd e' chiuso)

                        Cancella il file;
```

Nota1: posso creare un file temporaneo nel segg. modo:

fd=creat("temp",0666); unlink("temp");

Nota2: per creare semafori con i file posso fare cosi':

- creo un file senza permesso di scrittura
- uso la risorsa critica
- cancello il file con unlink

Ma: non funziona per il superuser!!

- APERTURA di file.

int open(char *path, int flags, int perms)

dove flags ha questi valori:

O_RDONLY	- lettura
O_WRONLY	- scrittura
O_RDWR	- lettura/scrittura
O_NDELAY	- file speciali
O_CREAT	- sostituisce **creat**
O_TRUNC	- se il file esiste, lo tronca a zero
O_EXCL	- assieme ad O_CREAT, fallisce se il file esiste gia'
O_APPEND	- scritture alla fine del file

Esempio: per aprire un file in scrittura se esiste, e di crearlo se non esiste:

```
fd=open(path, O_WRONLY|O_CREAT, 0666)
```

Per aprire il file in lettura e scrittura:

```
fd=open(path, O_RDWR|O_CREAT,0600)
```

Nota1: O_EXCL e' il modo ufficiale per usare un file come semaforo:

```
while((fd=open(path,O_WRONLY|O_CREAT|O_EXCL, 0666)) == -1)
```

funziona anche per il superuser.

Nota2: O_APPEND va' bene in caso di concorrenza. Non posso usare lseek e write per l'interleaving, a meno di non usare un semaforo

- SCRITTURA

int write(int fd, char *buf, unsigned nbytes)

Ritorna il numero di byte effettivamente scritti o -1.
Nota: la write non scrive effettivamente i dati su disco, ma segnala nel campo stato del relativo buffer nella buffer cache e ritorna! → scrittura ritardata

- LETTURA

int read(int fd, char *buf, unsigned nbytes)

Ritorna il numero di byte letti, 0 se EOF, -1 se errore.
Legge prima di tutto nella buffer cache.
Se il blocco di dati non esiste nella BC, avvia la lettura da disco. (vedi procedura bread)
La lettura e' molto veloce per multipli del blocco, perche' i dati vengono letti un blocco alla volta.

Per la lettura di caratteri e' conveniente usare le getc – putc dello stdio.h
Per leggere un numero di caratteri minore del blocco → user buffering

- CHIUSURA

int close(int fd)

rende disponibile il fd

- POSIZIONAMENTO

long lseek(int fd, long offset, int interp)

Ritorna il valore del file pointer oppure –1. Non fa' I/O ma modifica solo i valori in U-Area. Interpretazione di interp:

If(interp == 0)
    Offset assoluto;
else if (interp == 1)
    Offset relativo;
else if (interp == 2)
    L'offset viene sommato alla dimensione del file (usato per leggere all'indietro)

Uso di lseek:
- posizionamento assoluto
- posizionamento alla fine del file : lseek(fd, 0L, 2)
- per sapere dove ci si trova: lseek(fd, 0L, 1)

- PROPRIETA' DI UN DESCRITTORE

#include <fcntl.h>

int fcntl(int fd, int cmd, [int arg])

Cambia le proprieta' di un descrittore di un file aperto.
Interpretazione di cmd: 5 comandi, descritti in /usr/include/fcntl.h

F_DUPFD → ritorna un file duplicato >= arg (analogo a dup)
F_GETFD → ritorna un flag che indica se il file deve essere chiuso dopo una exec
F_SETFD → setta il flag di cui sopra
F_GETFL → ritorna i flag di stato del file fd (vedi open)
F_SETFL → setta i flag di stato del file ad arg

Esempio per settare il flag ‘append’:

```
int flags;
if (( flags = fcntl(fd, F_GETFL,0)) == -1 )
        printf(“Errore!\n”);
return(fcntl(fd,F_SETFL, flags|O_APPEND));
```

- INFORMAZIONI SUL FILE

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

int stat(char *path, struct stat *sbuf)
int fstat(int fd, struct stat *sbuf)
int lstat(const char *pathname, struct stat *sbuf)

stat e fstat leggono le informazioni da un I-node. Lstat come stat, ma se il file e’ un link simbolico, ritorna info sul link simbolico. I dati vengono messi nella struttura stat, definita in /usr/include/sys/stat.h.

```
struct stat {
        dev_t           st_dev;      //dev nr fs
        ino_t           st_ino;      //i-node nr
        mode_t          st_mode;     //file mode
        nlink_t         st_nlink;    //nr link
        uid_t           st_uid;      //owner uid
        gid_t           st_gid;      //owner gid
        off_t           st_size;     //size (byte)
        timestruc_t     st_atim;     //last acc
        timestruc_t     st_mtim;     //last mod.
        blksize_t       st_blksize;  //I/O block
        blkcnt_t        st_blocks;   //nr. blocks
};
```

- SET-UID/GID

  int setuid(int uid)

  int setgid(int gid)

Alzano I bit setuid/setgid per fare lavoro privilegiato in modo controllato. Ritornano –1 se errore.

- VERIFICA I PERMESSI AI FILE

int access(char *path, int pattern)

Ritorna 0 se i permessi indicati sono soddisfatti, -1 altrimenti.

- CAMBIA IL MODO

int chmod(char *path, int mode)

Cambia il modo di un file esistente. Coinvolge solo i 12 bit piu' a dx del mode.

- CAMBIA IL PROPRIETARIO

int chown(char *path, int owner, int group)

int fchown(int fd, int owner, int group)

int lchown(const char *path, int owner, int group)
//agisce sui link simbolici

path = file da modificare
owner = id del nuovo proprietario
group = id del nuovo gruppo

- COLLEGAMENTO

i**nt link(char *oldpath, char *newpath)**

Aggiunge un nuovo link in una directory. Il link ad una directory e' possibile solo da superuser.

i**nt symlink(char *oldpath, char *newpath)**

Crea un link simbolico.

- CAMBIA GLI ISTANTI DI ACCESSO

#include <sys/types.h>
int utime(char *path, struct utimebuf *timep)

dove:

```
struct utimebuf{
        time_t    atime;    // nuovo tempo d'accesso
        time_t    mtime;    // nuovo tempo di modifica
}
```

- CREAZIONE E RIMOZIONE DI DIRECTORIES

int mkdir(char *path, int mode) //alzare il bit 'x' per ricercare
//nella directory
int rmdir(char *path)

- CAMBIAMENTO DI DIRECTORIES

int chdir(char *path)

- CREA FILE SPECIALI

int mkfifo(char *path, int mode)

crea FIFO. Introdotto da Posix

int mknod(char *path, int mode, int device)

Crea un file di qualsiasi tipo. Sostituito da mkdir, mkfifo. I file ordinari sono creati con creat o open.
L'utente normale puo' usare mknod solo per fare FIFO.

```
int mkfifo(char *path)
{
    return(mknod(path, S_IFIFO|0666,0));
}
```