Lezione 6: FIle e Directory

6.1 Tipi di file
6.2 SC stat, fstat e lstat
6.3 Tipi di file
    6.3.1 Esempio
6.4 User ID e Group ID
6.5 set-user-id e set-group-id
6.6 Set user-ID
6.7 Set group ID
    6.7.1 Esempi: set UID e set GID
6.8 Permessi di accesso ai file
6.9 Sticky bit
6.10 Permessi di accesso ai file
6.11 Chiamata di sistema access
6.12 Proprietà dei nuovi file e directory
6.13 Permessi e creazione dei file
6.14 La system call umask
    6.14.1 Esempio
6.15 System call chmod e fchmod
    6.15.1 Esempio
6.16 System call chown, fchown d lchown
6.17 Dimensione del file
6.18 Troncamento
6.19 File System
6.20 Numero di link sulle directory
6.21 I-node
6.22 System call link
6.23 System call unlink
6.24 System call symlink e readlink
6.25 System call mkdir e rmdir
6.26 Lettura delle directory
    6.26.1 Lettura della directory (funzione libreria)
    6.26.2 Esempio: Elencari i file di una directory
6.27 System call chdir e funzione getcwd

6.27.1 Esempio
6.27.2 Esempio getcwd
6.28 Esercizio

6.1 Tipi di file

La maggior parte dei file in unix sono di due tipi regolari e directory. Esistono amnche tipi di file aggiuntivi.

I tipi di file sono:

- Regolari il più comune di file contenente dati in una qualche forma, per il kernel non c'é distinzione tra dati testo o binari.
- Directory contine nomi di altri file e puntatori alle informazioni di tali file.
- File speciali a blocco, sono usati per rappresentare dispositivi che consistono in un insieme di blocchi a indirizzamento casuale.
- File speciali a caratteri, son ousati per rappresentare dispositivi che consistono in un insieme di blocchi a indirizzamento casuale
- File speciali a caratteri, osno usati per rappresentare dispositivi che costituiscono flussi di caratteri.
- FIFO, usato per la comunicazione tra processi.
- Socket, tipo di file usato per la comunicazione su rete tra processi.
- Link simbolici.

6.2 SC stat, fstat e lstat

```
#include <sys/stat.h>
int stat (const char *path, struct stat *buf);
int fstat (int filedes, struct stat *buf);
int lstat (const char *path, struct stat *buf);
```

- stat, ritorna informazioni sul file specificato da path.
- fstat, ritorna informazioni sul file aperto sul descrittore filedes.
- lstat, ritorna informazionì sul link simbolico non sul file puntato da esso.

Il secondo argomento é un puntatore ad una struttura, le funzioni riempiono la struttura puntata da buf.

Queste funzioni ritornano informazioni sul file, non é necessario avere permessi di lettura sul file per accedere a queste informazioni. I permessi necessari sono di ricerca su tutte le directory nominate nel path.

6.3 Tipi di file

L'informazione sul tipo di file si trova nel campo st_mode della struttura stat per determinare il tipo di file si utilizzano le seguenti macro, definite in <sys/stat.h>

L'argomento di ciascuna macro è il campo st_mode.

S_ISLNK( )	symbolic link
S_ISREG( )	regular file
S_ISDIR( )	directory
S_ISCHR( )	character device
S_ISBLK( )	block device
S_ISFIFO( )	FIFO
S_ISSOCK( )	socket

6.3.1 Esempio

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

int main(int argc, char *argv[])
{
    int i;
    struct stat buf;
    char *ptr;

    for (i = 1; i < argc; i++) {
        printf("%s: ", argv[i]);
        if (lstat(argv[i], &buf) < 0) {
            printf("lstat error\n");
            continue;
        }
        if (S_ISREG(buf.st_mode)) ptr = "regular";
            …
        else if (S_ISBLK(buf.st_mode)) ptr = "block special";
        else ptr = "** unknown mode **";
        printf("%s\n", ptr);
}
exit(0);}
}
```

6.4 User ID e Group ID

Ogni file ha un proprietario ed un gruppo che lo possiede tali informazioni si trovano in st_uid e st_guid di stat

6.5 set-user-id e set-group-id

A ciascun processo vengono associati i seguenti identificativi:

- real user ID e reak group ID, identificano l’utente.
- effective user id e effective group id e supplementary group id, determinano i permessi di accesso ai file.
- saved set user id e saved set group id, contengono copia dell’effective user id e effective group id quando un programma è in esecuzione hanno un ruolo fondamentale per i file eseguibili.

Normalmente l’ EUID coincide con il RUID e EGID coincide con il RGIUD

6.6 Set user-ID

Un programma è eseguito con i permessi di chi lo manda in esecuzione, non di chi lo possiede.

Si può inizializzare un flag in st+mode in modo che quando un determinato file di programma viene eseguito l’effective user id del processo sia quello del proprietario del file.

Tale flag è detto set user id, potrebbe essere necessario che un processo abbia diritti maggiori di chi esegue il programma ad esempio con il comando passwd.

6.7 Set group ID

Analogamente é possivile imnpostare un altro bit in st_mode in modo che l’effective group id sia uguale al grupop proprietario del file il relativo flag é detto set group id.

Se il set groupo Id é applicato ad una directory i file creati in quella directory ereditano il group id dalla directory non dall’effective group id del processo che li ha crati.

6.7.1 Esempi: set UID e set GID

Sia exe il nome di un file eseguibile

Siano utente1 e gruppo1 i valori di UID e GID dell'utente che avvia exe, creando un processo P, che nel corso dell'esecuzione cercherà di accedere ad un file che chiameremo info.

Il real user ID di P è l' UID dell'utente che lo ha generato (utente1). Analogamente, il real
group ID di P è il GID dell'utente che lo ha generato (gruppo1)

L'effective user ID e l'effective group ID di P dipendono dai due bit speciali, set-user-id e etgroup-id, associati all'eseguibile exe

Se set-user-id è attivo, l'effective user id di P sarà uguale all'UID del proprietario del file
eseguibile in caso contrario l'effective user id di sarà uguale al suo real user id (cioè utente 1)Stesso discorso vale per set-group-i.

Vediamo come questi quattro identificatori sono utilizzati nel momento in cui il processo P cerca di accedere al file info (ricordiamo che info avrà associato un proprietario e un gruppo di utenti)

Valgono le seguenti regole (nell'ordine elencato):

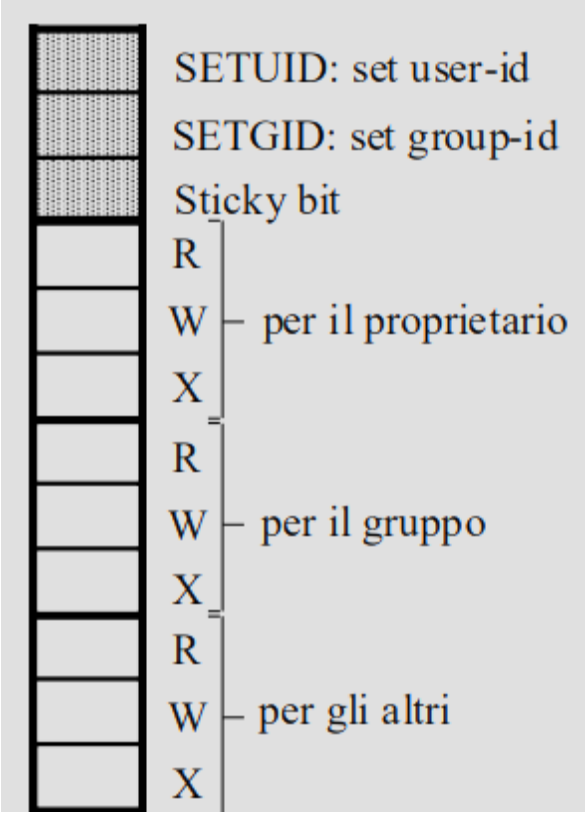
Se l'effective user id di P coincide con il proprietario di info, il processo acquisisce i diritti di accesso del proprietario di info

Altrimenti, se l'effective group id di P e il gruppo di info coincidono, P acquisisce i diritti di
accesso del gruppo di utenti associati ad info

Se nessuna delle due precedenti condizioni è valida, valgono le normali triple di diritti di accesso: l'accesso sarà consentito o meno a seconda della categoria di utenti nella quale ricadono real user id e real group id del processo

6.8 Permessi di accesso ai file

Complessivamente nel campo st_mode sono dedicati 12 bit peèi permessi.



Come si è visto st_mode contiene l'informazione relativa al tipo di file, il valore di st_mode codifica anche i bit di permesso di accesso ai file.

6.9 Sticky bit

Permette di richiedere al kernel che l'immagine del segmento di testo di un processo resti allocata nell'area di swap anche dopo la sua terminazione.

Se lo sticky bit é usato per la directory i file nella sua directory possono essere rinominati o cancellati solo se l'utente ha i permessi di scrittura sulla directory e se vale una delle seguenti.

- é il proprietario del file.
- é il proprietario della directory.
- é il superutente.

Usato per directory per cui un qualsiasi utente può creare file.

Gli utenti non dovrebbero avere la possibilità di rimnomincaere cancellarèfile di proprietà altrui.

6.10 Permessi di accesso ai file

Le tre categorie r w x sono usate da varie funzioni.

Quando si vuole aprire un qualsiasi tipo di file mediante il nome é necessario avere permessi di esecuzione x in ciascuna direcotry citata nel nome.

Ciò motiva il fatto che il bit del permesso di esecuzione per la directory è anche chiamato bit di ricerca.

I permessi di lettura determinano se possiamo aprire un file esistente per leggerlo.

I permessi di scrittura determinano se possiamo aprire un file per la scrittura.

è necessario avere permessi di scrittura su di un file per specificare il flag.

Non è possibile crearèun file in una directory a meno che non si hanno permessi di scrittura ed esecuzione sulla directory.

Per cancellare un file esistene è necessario avere i permessi di scrittura ed esecuzione nella directory che contiene il file.

6.11 Chiamata di sistema access

```
#include<unistd.h>
int access (const char *pathname, int mode)
```

access effettua il test di accessibilità di un file sulla base del real user ID a del real group ID.

Restituisce 0 se è ok altrimenti -1.

```
#include<fcntl.h>
int main(int argc, char*argv[]){
    if (argc!=2){
        printf(“usage: a.out <pathname>”);
        exit(-1);
    }
    if (access(argv[1], R_OK)<0)
        printf(“access error for %s”, argv[1]);
    else
        printf(“read access OK\n”);
    if (open(argv[1], O_RDONLY)<0)
        printf(“Open error for %s”, argv[1]);
    else
        printf(“open for reading OK\n”);
    exit(0);
}
```

6.12 Proprietà dei nuovi file e directory

Lo user ID di un nuovo file è impostato all’effective user ID del provesso. Per determinare il group ID, POSIX, consente alla implementazione di scegliere tra due opzioni:

1. il group ID di un nuovo file può essere l’effective group ID del processo.
2. il group ID del nuovo file può essere il group ID della directory i cui il file viene creato.

FreeBSD impiega sempre la seconda opzione.

Linux consente di scegliere l’una o l’altra sulla base di un flag impostato con mount.

6.13 Permessi e creazione dei file

```
#include <sys/stat.h>
mode_t umask(mode_t mask);
```

La SC umask viene utilizzata per assegnare ad un processo la modalità di creazione di un file.

L'argomento mask è formato da un OR bit a bit delle nuove costanti di permesso di accesso ai file.

La funzione ritorna il valore precedente della maschera di creazione dei file.

6.14 La system call umask

La maschera della modalità di creazione di un file è stata ogniqualvolta il processo crea un nuovo file o una nuova directory, i permessi di un file creato, dato un valore della maschera sono calcolati usando la seguente operazione bit a bit.

AND bit a bit tra il complemento di mask e la modalità di accesso specificata in creat o open.

6.14.1 Esempio

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include “apue.h"

int main(void) {
    umask(0);
    if (creat("foo", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP|S_IRO
        err_sys("creat error for foo");
        umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);
    if (creat("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_I
```

```
        err_sys("creat error for bar");
    exit(0);
}
```

6.15 System call chmod e fchmod

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

Queste funzioni permettono di cambiare i permessi di accesso ad un file.

Restituiscono 0 se è OK altrimenti -1

Per cambiare i bit di permesso di un file l'effective user ID del processo deve essere uguale all'ID del proprietario oppure il processo deve avere i diritti del superutente.

6.15.1 Esempio

```
#include <sys/types.h>
#include <sys/stat.h>
#include “apue.h"
int main(void)
{
 struct stat statbuf;
/* modo assoluto a "rw-r--r--" */
 if (chmod("bar", S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH) < 0)
 err_sys("chmod error for bar");
/* imposta set-group-ID disattiva group-execute */
 if (stat("foo", &statbuf) < 0)
 err_sys("stat error for foo");
 if (chmod("foo", (statbuf.st_mode & ~S_IXGRP) | S_ISGID) < 0)
 err_sys("chmod error for foo");
```

```
exit(0);
}
```

6.16 System call chown, fchown d lchown

```
#include <unistd.h>
int chown (const char *path, uid_t owner, gid_t group);
int fchown (int fd, uid_t owner, gid_t group);
int lchown (const char *path, uid_t owner, gid_t group);
```

Tali funzioni permettono di cambiare lo user ID ed il group ID di un file.

Se l'argomento owner o group è -1, l'ID corrispondente è lasciato inalterato. SOlo un processo superutente può modificare il proprietario. Un processo non superutente, proprietario del file può solo modificarne il gruppo con uno tra quelli supplementari a cui appartiene.

6.17 Dimensione del file

Il campo st_size di stat contiene la dimensione in byte del file. Ha senso solo per file regolari, directory e link simbolici.

Solari definisce anche la dimensione del file di una pipe come il numero di byte disponibili per lettura della pipe.

Se presenti st_blksixe e st_blocks si riferiscono, rispettivamente al migliore fattore di blocco per eseguire opererazioni di I/O sul file e al numero di blocchi da 512 byte allocati per il file.

La libreria standard del C utilizza tale fattore di blocco per eseguire le operazioni su file.

6.18 Troncamento

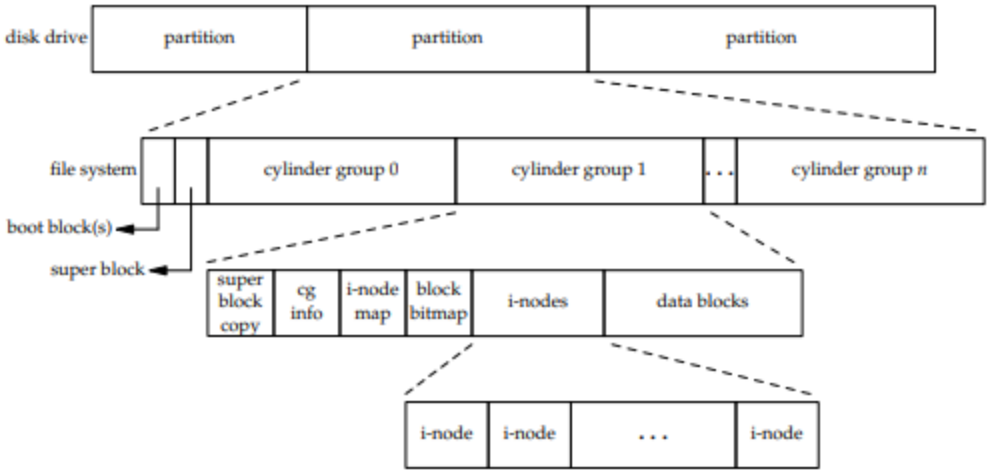
```
#include <unistd.h>
int truncate (const char *path, off_t length);
int ftruncate (int fd, off_t length);
```

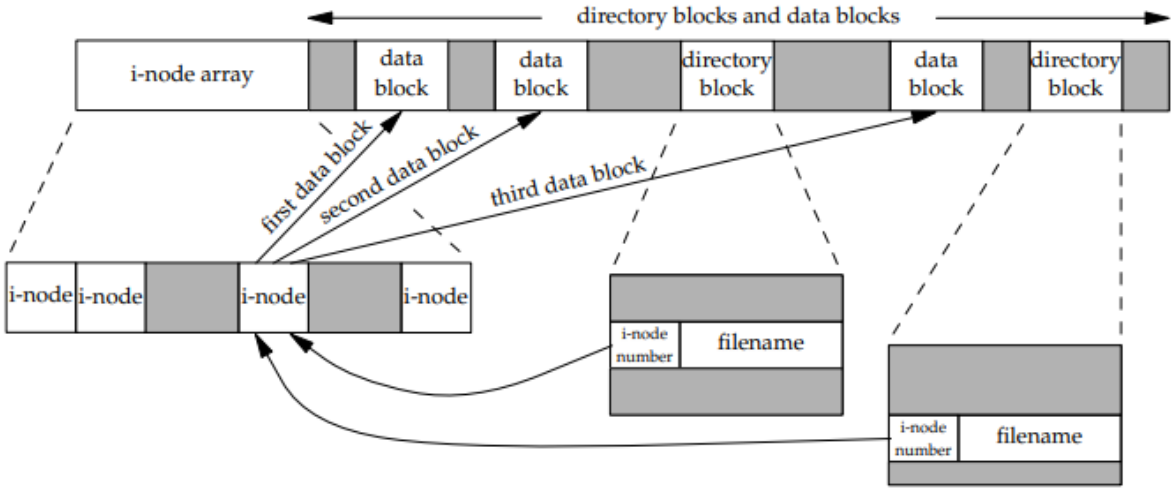
Queste SC troncano un file esistente a lenght byte.

Se il file ha una dimensione maggiore di lenght, i dati oltre length non sono più accessibili. Se il file ha meno di length byte, il comportamento della funzione dipende dall'implementazione.

6.19 File System

Consideriamo il file system di unix derivato da BSD possiamo pensare un HD suddiviso in una o più partizioni, ogni partizione può contenere un file system.





6.20 Numero di link sulle directory

Supponiamo di creare una directory testdire nella directory corrente, il numero di link in testdire è 2. Per l'i-node il numero di link è almeno 3.

6.21 I-node

Index-node è una struttura di controllo associata ad ogni file, molti nomi dei file possono essere associati con lo stesso i-node.

File Mode	16-bit flag that stores access and execution permissions associated with the file. 12-14 File type (regular, directory, character or block special, FIFO pipe 9-11 Execution flags 8 Owner read permission 7 Owner write permission 6 Owner execute permission 5 Group read permission 4 Group write permission 3 Group execute permission 2 Other read permission 1 Other write permission 0 Other execute permission
Link Count	Number of directory references to this inode
Owner ID	Individual owner of file
Group ID	Group owner associated with this file
File Size	Number of bytes in file
File Addresses	39 bytes of address information
Last Accessed	Time of last file access
Last Modified	Time of last file modification
Inode Modified	Time of last inode modification

6.22 System call link

```
#include <unistd.h>
int link (const char *oldpath, const char *newpath);
// Ritorna 0 se OK, -1 in caso di errore
```

è possibile far puntare più directory all' i-node di un file, la maniera per creare un hard link ad un file esistenziale è quella di usare la funzione link.

se newpart esiste già allora ritorna l'errore.

Se un'implementazione supporta la creazione di hard link su directory, il privilegio è ristretto al solo superutente

Il più delle implementazioni richiedono che oldpath e newpath risiedano nello stesso filesystem.

6.23 System call unlink

```
#include <unistd.h>
int unlink (const char *pathname);
// Ritorna 0 se OK, -1 in caso di errore
```

Per rimuovere un elemento dalla tabella della directory si utilizza la funzione unlink, questa funzione decremente il numero di link del file puntato da pathna,e, il file risulta ancora accessibile se il numero di linl è non nullo.

Quando il numero di link del file è 0 il contenuto del file può essere cancellato, ciò non accade se un processo ha il file aperto.

Quando il file viene chiuso, il kernel conta il numero di processi che hanno aperto il file: se questo è ed il numero di link del file è zero allora il file è cancellato.

è necessario avere i permessi di scrittura ed esecuzione nella directory contenente il file.

6.24 System call symlink e readlink

```
#include <unistd.h>
int symlink (const char *oldpath, const char *newpath);
```

```
int readlink (const char *path, char *buf, size_t bufsiz);
```

Symlink viene utilizzata per creare un link simbolico.

Per leggere un link simbolico è necessario utilizzare readlinl che apre il linl, legge il contenuto e lo chiude.

REstituisce il numero di byte letti se ok.

Il contenuto del link è posto in buf, senza carattere di terminazione.

6.25 System call mkdir e rmdir

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

```
#include <unistd.h>
int rmdir (const char *pathname);
```

Queste fnzioni permettono di creare directory e rimuoverle.

6.26 Lettura delle directory

Una directory può essere letta da chiunque abbia i permessi di accesso per lettura.

I permessi di esecuzione e scrittura per una directory determinano se si possono creare nuovi file nella directory e se si possono cancellare.

I permessi non specificano se si può scrivere sui file contenutinella directory stessa.

6.26.1 Lettura della directory (funzione libreria)

```
#include <dirent.h>
DIR *opendir(const char *name);
// Ritorna un puntatore se OK, NULL in caso di errore
struct dirent *readdir(DIR *dir);
// Ritorna un puntatore se OK, NULL alla fine o in caso di erro
void rewinddir(DIR *dir);
int closedir(DIR *dir);
```

6.26.2 Esempio: Elencari i file di una directory

```
#include <sys/types.h>
#include <dirent.h>
int main(int argc, char *argv[ ])
{
    DIR *dp;
    struct dirent *dirp;
    if (argc != 2){
        printf("a single argument (the directory name) is requi
         exit(-1);
     }

    if ( (dp = opendir(argv[1])) == NULL){
        printf("can't open %s", argv[1]);
        exit(-1);
    }

    while ( (dirp = readdir(dp)) != NULL)
        printf("%s\n", dirp->d_name);
        closedir(dp);
        exit(0);
}
```

6.27 System call chdir e funzione getcwd

```
#include <unistd.h>
int chdir (const char *path);
// Ritorna 0 se OK, -1 in caso di errore
char *getcwd (char *buf, size_t size);
// Ritorna puntatore a buf se OK,NULL in caso di errore
```

Ciascun processo è dotato di una directory di lavoro corrente da cui partono tutti i path relativi.

Per sapere qual è la directory corrente si usa getcwd

Per cambiare la directory di lavoro corrente di un processo si usa la funzione chdir

6.27.1 Esempio

```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    if (chdir(“/tmp”) < 0){
        perror("chdir fallito");
        exit(-1);
    } else
        printf("chdir a /tmp avvenuto\n");
    exit(0);
}
```

La directory di lavoro corrente è un attributo di un processo, I processi che invocano il processo che esegure chdir non sono influenzati dal cambio di

direcotory di quest’ultimo.

6.27.2 Esempio getcwd

```
#include “apue.h"
int main(void)
{
    char *ptr;
    int size;
    if (chdir("/home/staiano/bin") < 0)
        err_sys("chdir fallito");
        ptr = path_alloc(&size); /* funzione in apue.h */
    if (getcwd(ptr, size) == NULL)
        err_sys("getcwd fallita");
        printf("cwd = %s\n", ptr);
exit(0);
}
```

6.28 Esercizio

Esercizio per casa