

File e Directory

Date dei file

Ogni volta che accediamo ai dati contenuti in un file, il sistema memorizza la data di ultimo accesso

La stessa cosa accade quando modifichiamo i dati contenuti in un file o quando modifichiamo l'i-node del file

Queste informazioni sono memorizzate in tre campi della struttura **stat**

Date dei file

```
struct stat
```

```
{  
    dev_t          st_dev;          /* device */  
    ino_t          st_ino;         /* inode */  
    mode_t        st_mode;        /* file type & protection */  
    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 */  
};
```

Date dei file

Per ciascun file sono memorizzate tre date:

Campo	Descrizione	Esempio
st_atime	ultima data di accesso ai dati	read
st_mtime	ultima data di modifica dei dati	write
st_ctime	ultima data di modifica dell'i-node	chmod

Date dei file

Una modifica delle informazioni contenute nell'i-node (es. permessi del file) comporta una modifica di **st_ctime**, ma non di **st_mtime**

Il sistema non conserva la data di ultimo accesso all'i-node

Ciò significa che le operazioni di **stat** non modificano nessuna data

Funzione `utime`

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

```
#include <utime.h>
```

```
int utime (const char *filename, struct utimbuf *buf);
```

Per modificare la data di accesso e di modifica dei dati si utilizza **utime**

Funzione utime

La struttura utilizzata è

```
struct utimbuf {  
    time_t actime;    /* access time */  
    time_t modtime;  /* modification time */  
}
```

I due campi sono di tipo *calendar time*, (*numero di secondi a partire dalla mezzanotte del 1 gennaio 1970*)

Esempio

...

```
struct stat          statbuf;  
struct utimbuf      timebuf;
```

...

```
if (stat(filename, &statbuf) < 0)                                /* fetch current  
                                                                    times */
```

```
{ ... }
```

```
if (open(filename, O_RDWR | O_TRUNC) < 0) /* truncate */  
{ ... }
```

```
timebuf.actime    = statbuf.st_atime;
```

```
timebuf.modtime = statbuf.st_mtime;
```

```
if (utime(filename, &timebuf) < 0)                            /* reset times */  
{ ... }
```

...

Funzioni mkdir rmdir

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

int mkdir (const char *pathname, mode_t mode);
```

```
#include <unistd.h>

int rmdir (const char *pathname);
```

Queste funzioni permettono di **creare** directory e **rimuoverle** (*se vuote*)

Lettura delle directory

Una **directory** può essere **letta** da chiunque abbia i **permessi di accesso**

I **permessi di accesso** e **scrittura** per una **directory** determinano se si possono **scrivere** nuovi file nella **directory** e se si possono **cancellare**

I permessi non specificano se si può scrivere sui file contenuti nella **directory** stessa

Lettura delle directory

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

DIR *opendir(const char *name);
struct dirent *readdir(DIR *dir);
void rewinddir(DIR *dir);
int closedir(DIR *dir);
```

Il puntatore ad una struttura di tipo **DIR** ritornato da **opendir** è utilizzato dalle altre funzioni

```
struct dirent {
    ino_t d_ino;
    char d_name[NAME_MAX + 1];
};
```

Funzioni `chdir` `fchdir` `getcwd`

```
#include <unistd.h>

int chdir (const char *path);
int fchdir (int fd);
char *getcwd (char *buf, size_t size);
```

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

Per sapere qual'è la directory corrente si usa **getcwd**

Per cambiare la directory corrente di lavoro di un processo si usano le funzioni **chdir** e **fchdir**

Esempio

```
#include    <unistd.h>
#include    <stdio.h>
int main(int argc, char *argv[])
{
    int retval;
    if (chdir(argv[1]) < 0){
        perror("chdir failed");
        retval= -1;
    } else
    {
        printf("chdir to /tmp succeeded\n");
        retval= 0;
    }

    exit(retval);
}
```

Esempio

```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
    int retval;
    if (chdir(argv[1]) != 0)
        perror("chdir failed");
        retval = -1;
    } else
    {
        printf("chdir to %s succeeded\n", argv[1]);
        retval = 0;
    }
    exit(retval);
}
```

```
[mariog]$ a.out tmp
```

```
chdir failed: No such file or directory
```

```
[mariog]$ mkdir tmp
```

```
[mariog]$ a.out tmp
```

```
chdir to /tmp succeeded
```

```
[mariog]$ pwd
```

```
/users/mariog
```

```
[mariog]$
```

Esempio

```
#include      "ourhdr.h"

int main(void)
{
    char      *ptr;
    int       size;

    if (chdir("/users/mariog/bin") < 0)
        err_sys("chdir failed");

    ptr = path_alloc(&size);      /* our own function */
    if (getcwd(ptr, size) == NULL)
        err_sys("getcwd failed");

    printf("cwd = %s\n", ptr);
    exit(0);
}
```

Esempio

```
#include <sys/types.h>
#include <unistd.h>
#include "ourhdr.h"

int main(void)
{
    char *ptr;
    int size;

    if (chdir("/users/mariog/bin") != 0)
        err_sys("chdir failed");

    ptr = path_alloc(&size);
    if (getcwd(ptr, size) == NULL)
        err_sys("getcwd failed");

    printf("cwd = %s\n", ptr);
    exit(0);
}
```

```
[mariog]$ ln -s /usr/bin ./bin
```

```
[mariog]$ a.out
```

```
cwd = /usr/bin
```

```
[mariog]$
```

File speciali di dispositivo

Ciascun dispositivo in un sistema è rappresentato da un *file speciale di dispositivo* (*special device file*)

Ad esempio, il primo disco IDE in un sistema linux è rappresentato da **/dev/hda**

I driver identificano il dispositivo usando i *major e minor device number*

Tutti i dispositivi controllati dal kernel mediante uno stesso driver hanno in comune il major device number

I minor device number sono utilizzati per distinguere dispositivi (logici) differenti

Date dei file

struct stat

```
{
    dev_t          st_dev;          /* device */
    ino_t          st_ino;         /* inode */
    mode_t        st_mode;        /* file type & protection */
    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 */
};
```

File speciali di dispositivo

Ad esempio **ciascuna partizione** su di un disco ha un **diverso minor device number**, ma ha lo **stesso major number** delle altre partizioni.

Tali valori possono essere ottenuti tramite le macro **major** e **minor** dai membri della struttura **stat** di tipo **dev_t**

Il valore di **st_dev** di ciascun file è il numero di dispositivo e del filesystem a cui appartiene e che contiene il suo corrispondente i-node.

Solo i **file speciali a blocchi** e **a carattere** hanno un valore di **st_rdev**: questo valore è il numero relativo al **dispositivo fisico** su cui risiedono

```
#include <sys/types.h> /* BSD: defines major() and minor() */
#include <sys/stat.h>
#include "ourhdr.h"

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

    printf("%s: ", argv[1]);
    if (lstat(argv[1], &buf) < 0)
        perror("lstat error");

    printf("dev = %d/%d", major(buf.st_dev), minor(buf.st_dev));

    if (S_ISCHR(buf.st_mode) || S_ISBLK(buf.st_mode)) {
        printf(" (%s) rdev = %d/%d",
            (S_ISCHR(buf.st_mode)) ? "character" : "block",
            major(buf.st_rdev), minor(buf.st_rdev));
    }
    exit(0);
}
```

```
[mariog]$ a.out / ~ /dev/ttya[1-2]
```

```
/: dev = 3/1
```

```
/users/mariog: dev = 0/4
```

```
/dev/ttya1: dev = 3/1 (character) rdev = 3/177
```

```
/dev/ttya2: dev = 3/1 (character) rdev = 3/178
```

```
[mariog]$ df
```

Filesystem	1k-blocks	Used	Available	Use%
Mounted on				
/dev/hda1	995115	295679	648030	32% /
/dev/hda3	995147	766858	176881	82% /users

```
[mariog]$ ls -l /dev/hda[13] /dev/ttya[1-2]
```

```
brw-rw---- 1 root disk 3, 1 Aug 24 11:00 /dev/hda1
```

```
brw-rw---- 1 root disk 3, 3 Aug 24 11:00 /dev/hda3
```

```
crw-rw-rw- 1 root tty 3, 177 Aug 24 11:00 /dev/ttya1
```

```
crw-rw-rw- 1 root tty 3, 178 Aug 24 11:00 /dev/ttya2
```

```
[mariog]$
```

Funzioni **sync** **fsync**

Quando si esegue un'operazione di **write** su di un file, i dati vengono copiati dal kernel in un buffer e messi in coda per l'I/O

L'effettiva scrittura dei dati sul disco viene quindi postposta (*delayed write*)

Per assicurare la consistenza tra i dati sul disco ed il contenuto del buffer, sono fornite le funzioni **sync** ed **fsync**

Funzioni `sync` `fsync`

```
#include <unistd.h>
```

```
int sync (void);
```

```
int fsync (int fd);
```

La funzione **sync** mette in coda tutti i blocchi per la scrittura e ritorna

La funzione **fsync** agisce sul singolo file a cui si riferisce *fd*, aspettando che l'operazione di I/O sia completata prima di ritornare

Sebbene in alcune implementazioni **sync** effettivamente attende che l'operazione di I/O sia completata, ciò non assicura l'integrità dei dati
(*i dischi sono dotati di cache*)

Sommario

Abbiamo analizzato tutti gli attributi di un file Unix

La conoscenza delle proprietà dei file e delle funzioni che operano su di essi è fondamentale nella programmazione del sistema Unix

Esercizi

Aggiungere l'opzione `-r` (ricorsione) al comando **mycat** implementato

Es.

```
$ mycat -r +10 *.c
```