Directory

Le **directory** unix sono **file**.

Molte system call per i file ordinari possono essere utilizzate per le directory. E.g. `open`, `read`, `fstat`, `close`.

Tuttavia le directory non possono essere create con `open`, `creat`.

Esiste un insieme di system call speciali per le directory.

Una directory è rappresentata in memoria da una **tabella**, con una entry per ogni file nella directory.

inode	name
⋮	⋮

Ogni entry è una struttura di tipo `dirent` definita in `<dirent.h>`:

```
ino_t d_ino;
char d_name[ ];
```

Il primo campo contiene l'inode del file, il secondo il nome del file. Se `d_ino`=0, allora lo slot è libero.

Creazione e apertura di una directory

Creazione di una directory:

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

La system call `mkdir` restituisce 0 o -1 a seconda che termini con successo o meno. Al momento della creazione con `mkdir`, i link . e .. vengono inseriti nella tabella.

Apertura di una directory:

```
#include <sys/types.h>
#include <dirent.h>
DIR *opendir (const char *dirname);
```

La system call `opendir` ritorna un puntatore di tipo `DIR` oppure il null pointer, in caso di fallimento.

Lettura, riposizionamento, chiusura di una directory

Lettura di una directory:

```
#include <sys/types.h>
#include <dirent.h>
struct dirent *readdir (DIR *dirptr);
```

La system call `readdir` restituisce un puntatore alla struttura `dirent` dove è stata copiata la prossima entry nella tabella.

Riposizionamento:

```
void rewinddir (DIR *dirptr);
```

Chiusura:

```
#include <dirent.h>
int closedir (DIR *dirptr);
```

Esempio: lettura di una directory

```
#include <dirent.h>

int my_ls (const char *name)
{
  struct dirent *d;
  DIR *dp;

  /* apertura della directory */
  if ((dp=opendir(name)) == NULL)
     return (-1);

  /* stampa dei nomi dei file contenuti nella directory */
  while (d = readdir(dp))
  {
     if (d->d_ino != 0)
        printf("%s\n", d->d_name);
  }

  closedir(dp);
  return(0);
}
```

Tipo di un file

Come si è visto nella lezione precedente, la system call `stat` serve per accedere ad una struttura dati contenente gli attributi di un file.

L'attributo `st_mode` contiene il **file mode**, cioè una sequenza di bit ottenuti facendo l'OR bit a bit della stringa che esprime i permessi al file e una costante che determina il tipo del file (regolare, directory, speciale, etc.).

Per scoprire se un file è una directory, si può usare la macro `S_ISDIR`:

```
/* buf e' il puntatore restituito da stat */
if (S_ISDIR (buf.st_mode))
    printf("It is a directory\n");
  else
    printf("It is not a directory\n");
```

Cambiamento della directory corrente

La directory corrente di un processo è quella in cui il processo è eseguito. Tuttavia un processo può cambiare la sua directory corrente con la system call

```
#include <unistd.h>
int chdir (const char *path);
```

dove `path` è il pathname della nuova directory corrente. Il cambiamento si applica solo al processo chiamante.

Attraversamento dell'albero di una directory

La system call `ftw` consente di eseguire un'operazione `func` su tutti i file nella gerarchia della directory `path`:

```
#include <ftw.h>
int ftw (const char *path, int (*func)(), int depth);
```

Il parametro `depth` controlla il numero di file descriptor usati da `ftw`. Più grande è il valore di `depth`, meno directory devono essere riaperte, incrementando la velocità di esecuzione.

`func` è una funzione definita dall'utente, che viene passata alla routine `ftw` come puntatore a funzione. Ad ogni chiamata, `func` viene chiamata con tre argomenti: una stringa contenente il nome del file a cui `func` si applica, un puntatore ad una struttura `stat` con i dati del file, un codice intero. Il prototipo di `func` deve perciò essere:

```
int func (const char *name, const struct stat *sptr, int type);
```

L'argomento `type` contiene uno dei seguenti valori (definiti in `<ftw.h>`), che descrivono il file oggetto:

`FTW_F`	l'oggetto è un file
`FTW_D`	l'oggetto è una directory
`FTW_DNR`	l'oggetto è una directory che non può essere letta
`FTW_SL`	l'oggetto è un link simbolico
`FTW_NS`	l'oggetto non è un link simbolico e su di esso `stat` fallisce

Esempio (I)

Il programma `rls` (recursive ls) prende come argomento sulla linea di comando una directory e visualizza su standard output tutti i file regolari e le directory che incontra attraversando il file system a partire da quest'ultima (evidenziando se si tratta di file ordinari o di directory).

```
#include <ftw.h>
#include <stdio.h>
#include <string.h>

int src(const char *name,const struct stat *sptr,int type);

main(int argc,char **argv) {

    if(argc!=2) {
        fprintf(stderr,"Utilizzo: rls <dir>\n");
        exit(1);
    }
```

Esempio (II)

```
    if(ftw(argv[1],src,5)==-1) {
        perror("Errore nell'esecuzione di ftw");
        exit(2);
    }

}

int src(const char *name,const struct stat *sptr,int type) {

    if(type==FTW_F || type==FTW_D) {
        printf("%s ",name);

        if(type==FTW_F)
            printf("(file ordinario)\n");
        else
            printf("(directory)\n");

    }

    return 0;
}
```

Esercizio

Si scriva un programma C che realizza una versione semplificata del comando unix `find`. Il programma dovrà ricevere sulla linea di comando il nome di una directory *dir* ed una stringa *str* e dovrà visitare l'intero albero di directory e file che ha come radice *dir*, stampando su std output tutti i file i cui nomi hanno come suffisso la stringa *str*, segnalando se si tratta di directory o file ordinari.