DIRECTORY

➔ trattamento directory a livello di programmazione

Ogni directory viene considerato da UNIX (e MS-DOS) come un file con caratteristiche speciali ⇨ file che contiene descrittori di altri file (*file speciale*)

Da un punto di vista astratto, si potrebbe quindi pensare di usare le stesse funzioni utilizzabili per i file *normali*: in particolare, le operazioni open() e read()
⇨ necessità di conoscere nei dettagli la struttura interna del file-directory

Sistema Operativo impedisce l'accesso a tali file con le funzioni utilizzabili sui file *normali* ⇨ funzioni primitive *ad-hoc* che consentono l'accesso ai directory prescindendo dalla loro effettiva implementazione

VANTAGGI: i programmi che usano queste funzioni risultano essere portabili su qualunque Sistema Operativo che realizzi tali funzioni

Oltre alle funzioni per l'accesso ai directory, deve essere prevista anche una funzione che modifica il directory corrente ed una funzione di creazione

PRIMITIVE CHE AGISCONO SUI DIRECTORY

1) CREAZIONE DI UN DIRECTORY

MKDIR

```
retval = mkdir (pathname, mode);
        char * pathname;
        int mode;
/* diritti sul directory */
        int retval;
```

La primitiva **MKDIR** crea un directory con il nome e i diritti specificati ⇨ vengono sempre creati i file

- **.** (link al directory corrente)
- **..** (link al directory padre)

mkdir restituisce il valore 0 in caso di successo, altrimenti un valore negativo

2) CAMBIO DI DIRECTORY

```
retval =   chdir (nomedir);
   char *nomedir;
   int retval;
```

La primitiva **CHDIR** consente di cambiare il directory corrente

Questa funzione **restituisce 0** se **successo** (cioè il cambio di directory è avvenuto),
altrimenti **restituisce -1** (in caso di **insuccesso**)

segue PRIMITIVE SUI DIRECTORY

3) APERTURA DI DIRECTORY

```
dir  =  opendir (nomedir);
    char *nomedir;
    DIR *dir;
/* DIR è una struttura astratta e non usabile dall'utente
*/
```

La primitiva **OPENDIR** consente di aprire una sessione di accesso ad un directory

Questa funzione **restituisce** un valore diverso da **NULL** se ha **successo** (cioè l'apertura del directory è avvenuta), altrimenti **restituisce NULL** (in caso di **insuccesso**)

4) CHIUSURA DI DIRECTORY

```
retval =  closedir (dir);
    DIR *dir;
    int retval;
```

La primitiva **CLOSEDIR** effettua la chiusura del directory

Questa funzione **restituisce 0** se **successo** (cioè la chiusura del directory è avvenuto),
altrimenti **restituisce -1** (in caso di **insuccesso**)

segue PRIMITIVE SUI DIRECTORY

5) LETTURA DI DIRECTORY

```
descr = readdir (dir);
    DIR *dir;
    struct dirent *descr;
```

La funzione **restituisce** un valore diverso da **NULL** se ha avuto **successo** (cioè la lettura del directory è avvenuta), altrimenti **restituisce NULL** (in caso di **insuccesso**)
In caso di successo, descr punta ad una struttura di tipo dirent
Ad esempio:

```
struct  dirent   {
    long              d_ino;     /* i_number */
    off_t             d_off;     /* offset del prossimo */
    unsigned short    d_reclen;  /* lunghezza del record */
    unsigned short    d_namelen;/* lunghezza del nome */
    char              d_name[1];/* nome del file */
              }
```

la stringa che parte da descr -> d_name rappresenta il nome di un file nel directory aperto

Questa stringa termina con un carattere nullo (convenzione C) ⇨ possibilità di nomi con lunghezza variabile
La lunghezza del nome è data dal valore di d_namelen

In MS-DOS:

```
struct dirent
{ char        d_name[13]; };
```

NOTA BENE:
Le primitive **chdir**, **opendir**, **readdir** e **closedir** sono **INDIPENDENTI** dalla specifica struttura interna del directory
➔ valgono sia per Unix (BSD o System V) che per MS-DOS (libreria BorlandC)

ESEMPIO:

Implementazione (semplificata) del comando ls di UNIX o DIR di MS-DOS

```
#include <sys/types.h>  /* per UNIX */
#include <dirent.h>
#include <fcntl.h>

my_dir (char *name)
{   DIR *dir;
    struct dirent *f;
    int count = 0;

    dir = opendir (name);
    if (dir == NULL)
           { puts("ERRORE"); return 2; }
    while ((f = readdir(dir)) != NULL)
        {   printf("Trovato il file %s\n", f -> d_name);
            count++;
        }
    printf("Numero totale di file %d\n", count);
    closedir (dir);
    return 0;
}

main (int argc, char *argv[ ])
{ int status;
    if (argc != 2)
        { printf("Errore nel numero di parametri\n");
          exit(1); }
    printf("Esecuzione di mydir\n");
    status = my_dir(argv[1]);
    if (status != 0)
        printf ("Ci sono stati degli errori in mydir\n");
    exit (status);
}
```

ESEMPIO:

Si vuole operare su una gerarchia di DIRECTORY alla ricerca di un file con nome specificato

Per ESPLORARE la gerarchia si utilizza la funzione per cambiare directory **chdir** e le funzioni **opendir**, **readdir** e **closedir**

```
/* file dirfun.c */
#define NULL 0
#include <sys/types.h>
#include <dirent.h>

/* La soluzione seguente ASSUME che il nome del directory
sia dato in modo ASSOLUTO.
NOTA BENE: questa soluzione va bene se e solo se il
directory di partenza non è la radice (/). PERCHÈ ? */

void esplora (char *d, char *f);

main (int argc,char ** argv)
{
    if (argc != 3)
    { printf (" errore\n "); exit (-1); }
    if (chdir (argv[1]))  exit(-2);
    esplora (argv[1], argv[2]);
}
```

```
/* funzione di esplorazione di una gerarchia: opera in modo RICORSIVO */
void esplora (char *d, char *f)
{   char nd [80];
    DIR  *dir;
    struct dirent *ff;

    dir = opendir(d);
    if (dir == NULL) { puts("ERRORE"); exit(-1); }
    while (((ff = readdir(dir)) != NULL))
    {   if ( (strcmp (ff->d_name, ".") == 0) ||
            (strcmp (ff->d_name, "..") == 0)) continue;
/* bisogna saltare i nomi del directory corrente e del directory padre */
        if (chdir(ff -> d_name) != 0)
/* è un file e non un directory */
        {   if ( strcmp ( f, ff-> d_name)  == 0)
            printf("Trovato il file %s nel directory %s\n", f, d);
/* eventuali altri operazioni sul file:
        ad esempio apertura,etc. */
        }
        else  /* trovato un directory */
            {   strcpy(nd, d); strcat(nd, "/");
            strcat(nd, ff-> d_name);
/* costruzione nome assoluto */
            esplora ( nd, f);
            chdir("..");
/* bisogna tornare su di un livello per ritrovare i nomi dei file */
            }
    }
    closedir(dir);
}
```

ESEMPIO (seconda soluzione):

Si vuole operare ISOLANDO le primitive di accesso ai directory

Questa soluzione prevede la presenza di una funzione di nome **findfiles** il cui prototipo (che supponiamo si trovi nel file *findfile.h*) è:

```
/* file findfile.h */
extern char ** findfiles (char *d);
/* restituisce l'insieme dei nomi dei file (e directory)
del directory specificato ===> nasconde i dettagli
relativi alla apertura, lettura e chiusura del
file/directory */
```

NOTA BENE: Il formato del valore ritornato da questa funzione è lo stesso di argv ed envp

```
/* file dirfun.c */
/* La soluzione seguente ASSUME che il nome del directory
di partenza della GERARCHIA sia dato in modo ASSOLUTO */
#include "findfile.h"

void esplora(char *d, char *f);

main (int argc, char **argv)
{
    if (argc != 3)
        { printf ("Errore nel numero di parametri\n");
exit (-1); }
    if (chdir (argv[1]))  exit(-2);
    esplora (argv[1], argv[2]);
}
```

```
/* funzione di esplorazione di una gerarchia: opera in
modo RICORSIVO */

void esplora (char *d, char *f)
{   char nd [80]; char **nf;
  nf = findfiles(d);
/* restituisce la lista dei nomi dei file (e directory)
del directory specificato */
  while (*nf)
   { if (chdir(*nf) != 0)
/* è un file e non un directory */
     { if ( strcmp ( f, *nf)  == 0)
           printf("Trovato il file %s nel directory %s\n", f, d);
     }
  else
/* directory */
     { strcpy(nd, d); strcat(nd, "/");
           strcat(nd, *nf); /* nome assoluto */
       esplora(nd, f);  chdir("..");
     }
  nf++;
   }
}
```

```
/* file findfile.c */
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#define NULL 0

char ** findfiles (char *d)
{   int nr, j, i=0;
    struct dirent *f;   /*   puntatore   alla   struttura
relativa al nome di file/directory corrente */
    DIR *dir;  /*  puntatore  alla  struttura  relativa  al
directory aperto */
    char **reply; /* puntatore alla struttura dinamica che
rappresentera' il valore di ritorno della funzione */

reply = (char **) malloc(200 * sizeof(char *));
/*  si  allocano  200  locazioni  ognuna  per  contenere  un
puntatore a carattere: si ipotizza quindi al massimo di
avere 200 nomi di file/directory per ogni directory */
dir = opendir(d);    /* apertura del directory */
if (dir == NULL) { puts("ERRORE"); exit(-1); }
/* controllo sull'apertura */
while ((f = readdir(dir)) != NULL)
/* leggiamo, fino alla fine del file/directory */
    { if ( !(strcmp (f -> d_name, ".") == 0) &&
           ! (strcmp (f -> d_name, "..") == 0) )
/*  si  decide  di  saltare  i  file  speciali  (directory
corrente e directory padre) */
       { reply[i]=(char *) malloc(strlen(f->d_name) + 1);
       strcpy(reply[i], f -> d_name);
    /* per ogni nome trovato, si crea un elemento nella
lista reply e vi si copia tale nome */
       i++; }
    }
reply[i] = NULL; /*  terminiamo  la  lista  reply  con  la
stringa nulla */
closedir (dir);  /* chiudiamo il directory */
return reply; /* ritorniamo la lista di nomi */
}
```

NOTA BENE: Si è deciso di eliminare dalla lista tornata dalla findfiles, i nomi dei file speciali che consentono di identificare il directory corrente (.) ed il directory padre (..)

ALTRO ESEMPIO:

Consideriamo il problema dell'esplorazione ricorsiva di una gerarchia il cui nome assoluto viene dato come parametro al programma per riportare la struttura della gerarchia in modo simile al comando **tree** di MS-DOS

```
/* TREE: visualizza il nome di tutti i file e directory
di una gerarchia data */
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>
#include <string.h>
#ifdef MS_DOS
#define SEP '\\'
#define SEPSTR "\\"
#else
#define SEP '/'
#define SEPSTR "/"
#endif

#include "findfile.h"
/* extern char ** findfiles (char *d); */

void listadirfile (char *);

int main  (int argc, char **argv)
{/* controllo numero parametri */
if (argc != 2)
   { printf ("Errore: numero di parametri errato\n ");
     exit (-1); }
if (argv[1][0] != SEP)
/* controllo se il nome del directory viene dato in forma
assoluta */
   { printf ("Errore: il primo parametro non e' un nome
assoluto\n "); exit (-2); }
/* controllo se il primo parametro e' un directory */
if (chdir (argv[1]))
   { printf ("Errore: il primo parametro non e' un
directory\n "); exit (-4); }
/* chiamata funzione che esplora ricorsivamente la
gerarchia che inizia dal directory argv[1]: nota siamo
gia' nel directory giusto */
listadirfile(argv[1]);
}
```

segue ESEMPIO

```
void listadirfile(char *d)
/* funzione ricorsiva che esplora la gerarchia data */
{   char nd [80];
    char **nf;

nf = findfiles(d);
while (*nf)   /* scorriamo la lista fino a che ci sono
nomi di file e directory */
    { if (chdir(*nf) == 0)
    /* abbiamo trovato un directory */
        { strcpy(nd, d);
         if (nd[strlen(nd) - 1] != SEP)
                              strcat(nd, SEPSTR);
         strcat(nd, *nf);
    /* costruzione del nome assoluto del directory */
        printf( "Directory: %s\n", nd);
    /* visualizziamo il nome del directory */
        listadirfile(nd);    /* invocazione ricorsiva */
        chdir(".."); /* torniamo al livello precedente */
         }
        else
/*  abbiamo trovato un file: visualizziamo il suo nome
completo  */
    printf( "File: %s"SEPSTR"%s\n", d, *nf);
    /* OPPURE; printf("File: %s", d); printf(SEPSTR);
                printf("%s\n", *nf); */
     nf++;
    /* passiamo ad un altro elemento della lista */
    }
}
```

segue ESEMPIO

OSSERVAZIONI:

1. Questa soluzione tiene conto della portabilità ⇨ versione che va bene sia per UNIX che per MS-DOS poiché considera che è necessario un diverso carattere per separare i nomi dei sottodirectory nei nomi assoluti dei file e dei directory. Nel caso del Sistema Operativo MS-DOS dato che il carattere da usare è il carattere di back-slash (\), è necessario in C scrivere \\: questo perché in C il simbolo \ ha il significato di carattere di *escape*

2. Si ricorda che in MS-DOS i nomi di file e di directory possono essere scritti indifferentemente con caratteri maiuscoli o minuscoli, contrariamente a UNIX che è, invece, *case-sensitive*
   Nel caso, però, si intenda effettuare la ricerca di un file o di un directory con un nome specifico, utilizzando, ad esempio, la funzione strcmp() si segnala che la stringa da cercare deve essere data utilizzando caratteri tutti maiuscoli poiché il Sistema Operativo memorizza in tale formato i nomi dei file/directory

3. Il controllo se il nome corrente (*nf) è quello di un file o di un directory viene fatto con la primitiva chdir(): se questa ha successo, vuol dire che il nome è quello di un directory, altrimenti il nome è quello di un file
   Notiamo che nel primo caso, la primitiva ha come effetto collaterale quello di modificare il directory corrente
   Per tale ragione, dopo avere effettuato la invocazione ricorsiva, effettuiamo una chiamata alla primitiva chdir("..") per tornare al livello precedente