13.7 Funzioni di basso livello

Alcune versioni del C offrono un altro insieme di funzioni per operare sui file. La modalità d'uso di queste funzioni è assai simile a quella delle funzioni precedenti, ma il loro impiego è limitato allo sviluppo di applicazioni che abbiano la necessità di raggiungere notevoli prestazioni. Queste funzioni sono di più basso livello rispetto alle precedenti e corrispondono spesso direttamente alle chiamate al sistema operativo. Il lettore, se lo desidera, può saltare questo

paragrafo a una prima lettura.
Il programmatore deve prestare molta attenzione a non utilizzare contemporaneamente le due classi di funzioni sul medesimo file; infatti le due strategie interne di gestione dei file sono differenti e l'uso contemporaneo delle due classi di funzioni può generare errori ed effetti collaterali all'interno del programma la cui portata non è valutabile a priori.
Le funzioni precedenti utilizzavano il concetto di file pointer per operare sui file, quelle che andiamo a descrivere ora implementano un concetto analogo, che prende il nome di *file descriptor*, talvolta chiamato "maniglia" o "canale". Il file descriptor è un numero intero che viene associato dalla funzione di apertura al file sul quale si desidera operare.
Per lavorare con queste funzioni di accesso ai file è necessario includere: `fcntl.h`, `sys/types.h` e `sys/stat.h`.
La funzione per aprire un file si chiama `open` e lavora come nell'esempio seguente:

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

main()
{
int fd;
fd = open("clienti", O_RDONLY);
...
```

Tale funzione associa il file descriptor `fd` al file di nome `clienti` aprendolo in modalità di sola lettura. Il valore di ritorno della `open` può essere negativo nel caso in cui si sia verificato un errore, per esempio se il file `clienti` non esiste. L'uso generale della funzione `open` su `clienti` è:

```
int fd, modo, diritti;
...
fd = open("clienti", modo [diritti] );
```

dove *modo* rappresenta la modalità di apertura del file è può avere il valore di una o più delle seguenti costanti simboliche:

`O_RDONLY`	apre il file in sola lettura; se il file non esiste la `open` ritorna errore
`O_WRONLY`	apre in file in sola scrittura; se il file non esiste la `open` ritorna errore
`O_RDWR`	apre il file in lettura e scrittura; se il file non esiste la `open` ritorna errore
`O_CREAT`	crea il file se non esiste
`O_TRUNC`	distrugge il contenuto del file preesistente
`O_APPEND`	apre il file in modalità append; tutte le scritture sono automaticamente eseguite a fine file
`O_EXCL`	la `open` ritorna errore se il file esiste già al momento dell'apertura

Se il programmatore desidera specificare più di una modalità di apertura lo può fare utilizzando l'operatore binario di OR bit a bit; per esempio

```
fd = open("clienti", O_WRONLY | O_TRUNC);
```

fornisce l'apertura del file `clienti` in scrittura con distruzione del contenuto preesistente.

Nel caso in cui sia stata specificata la modalità `O_CREAT` il programmatore deve anche specificare i `diritti` o permessi con i quali il file deve essere creato; tali permessi sono solitamente codificati con una sintassi simile a quella utilizzata dal sistema operativo Unix, in cui chi utilizza un file rientra sempre in almeno una di queste categorie:

- è il possessore del file;
- appartiene al gruppo collegato al file;
- non è collegato in alcun modo al file.

Per ciascuna categoria, è possibile specificare i permessi che consentono l'utilizzazione del file mediante la forma ottale, che è costituita da tre o quattro cifre comprese tra 0 e 7, per esempio:

`0640`

In questo contesto tralasciamo il significato della prima cifra a sinistra che è opzionale. Ogni cifra viene interpretata come una somma delle prime 3 potenze di 2 (2^0=1, 2^1=2, 2^2=4), ciascuna delle quali corrisponde a un determinato tipo di permesso – dove la seconda rappresenta il proprietario, la terza il gruppo e l'ultima a destra tutti gli altri utenti –; la corrispondenza è la seguente:

`4` permesso di lettura
`2` permesso di scrittura
`1` permesso di esecuzione
`0` nessun permesso

Dunque con `640` abbiamo i permessi di lettura e scrittura per il proprietario (`4` permesso di lettura + `2` permesso di scrittura), di sola lettura per il gruppo e nessun permesso agli altri.

Vediamo un esempio dove definiamo più di una modalità e anche i `diritti`: il codice

```
int fd;
fd = open("clienti", O_RDWR | O_CREAT | O_TRUNC, 0640);
```

crea il file `clienti` aprendolo in lettura e scrittura con diritti di lettura/scrittura per il proprietario (`6`), con soltanto diritto di lettura per il gruppo di utenti cui appartiene il proprietario (`4`) e con nessun diritto per tutti gli altri utenti (`0`).
La funzione `close` ha un comportamento analogo a quello della funzione `fclose`: chiude un file descriptor aperto dalla `open`:

```
int fd;
...
close(fd);
```

Per quanto riguarda le operazioni di lettura e scrittura su file con utilizzo dei file descriptor, le funzioni che le eseguono si chiamano rispettivamente `read` e `write`.
La funzione `read` opera nel modo seguente:

```
char buf[1000];
int elementi;
int fd;
int n;
...
elementi = 1000;
n = read(fd, buf, elementi);
```

dove `fd` è il file descriptor da cui si desidera leggere, `buf` è il vettore dove i dati letti devono essere trasferiti ed `elementi` rappresenta la dimensione in byte del vettore. Il valore di ritorno della funzione indica quanti byte sono stati realmente letti dal file `fd`; tale valore può essere inferiore alla dimensione di `buf` nel caso in cui il puntatore al file abbia raggiunto la fine; un valore di ritorno uguale a zero indica che siamo giunti a fine file.
La funzione `fwrite` lavora così:

```
char buf[1000];
int elementi = 1000;
int n, fd;
...
n = write(fd, buf, elementi);
```

dove `fd` è il file dove si desidera scrivere, `buf` contiene i dati che devono essere scritti ed `elementi` rappresenta il numero di byte da scrivere. Il valore di ritorno della `write` indica il numero di byte scritti sul file; tale valore può essere inferiore a `elementi` nel caso in cui il file abbia superato la massima dimensione ammessa.
Scriviamo un programma di esempio che copia il contenuto di un file in un altro utilizzando i file descriptor (Listato 13.8).

✓ **NOTA**
Questa volta abbiamo utilizzato la funzione `exit` che fa terminare il programma (si veda il Paragrafo 3.8). I programmatori C usano spesso questa possibilità del linguaggio, che noi abbiamo circoscritto agli esempi di questo paragrafo. Se non si ritiene opportuno questo approccio è sufficiente togliere la `exit` e aggiungere un ramo `else` a `if(fp==NULL)`, con un blocco dove raccogliere la parte seguente del programma.

```
/* Copia il contenuto di un file in un altro */

#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

main(argc,argv)
int argc;
char **argv;
{
static char buf[BUFSIZ];
int fdin, fdout, n;

if( argc != 3 ) {
   printf("Devi specificare file sorgente e destinazione\n");
   exit(1);
}

/* Apre il file sorgente */
fdin = open(argv[1],O_RDONLY);
if( fdin < 0 ) {
   printf("Non posso aprire %s\n",argv[1]);
   exit(2);
}

/* Apre il file destinazione */
fdout = open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0600);
if( fdout < 0 ) {
   printf("Non posso creare %s\n", argv[2]);
   close(fdin);
   exit(3);
}

/* Esegue il ciclo di lettura e scrittura */
for(;;) {
   /* Legge BUFSIZ caratteri da file */
   n = read(fdin, buf, BUFSIZ);

   /* Controlla la fine del file */
   if( n == 0 )
      break;

   /* Scrive i caratteri nel file destinazione */
   write(fdout,buf,n);
}

/* Chiude i file */
close(fdin);
close(fdout);
```

```
}
```

Listato 13.8 Programma che copia il contenuto di un file in un altro utilizzando i file descriptor

Abbiamo visto in precedenza come sia possibile spostare il file pointer utilizzando la funzione `fseek`; anche per i file descriptor esiste una funzione, `lseek`, che consente di muovere il puntatore al file:

```
long offset;
long n;
int mode;
int fd;
...
offset = lseek(fd, n, mode);
```

dove `fd` è il file descriptor sul quale si desidera muovere il puntatore, `n` rappresenta il numero di byte di spostamento. Se `n` è negativo lo spostamento del puntatore avviene all'indietro invece che in avanti. Il parametro `mode` indica a partire da quale posizione iniziare a muovere il puntatore: se `mode` vale `0` significa che ci si deve muovere a partire dall'inizio del file, se vale `1` a partire dalla posizione corrente e infine se vale `2` a partire dalla fine. Il valore di ritorno della `lseek` contiene la posizione corrente del puntatore dopo lo spostamento. Allora:

`lseek(fd,0L,1)`	restituisce la posizione corrente
`lseek(fd,0L,2)`	restituisce la dimensione del file in byte

Utilizzando i file pointer, avevamo visto come il sistema offriva tre file pointer automaticamente aperti al momento del lancio del programma (`stdin, stdout, stderr`) mediante i quali era possibile lavorare su tastiera e video. Anche usando i file descriptor il sistema mette a disposizione tre descrittori aperti per default al momento del lancio del programma e associati a tastiera e video:

`0`	standard input associato a tastiera
`1`	standard output associato al video
`2`	standard error associato al video

Questi tre numeri interi possono essere utilizzati dal programmatore per leggere e scrivere da tastiera e video senza dover eseguire le relative `open`. Scriviamo dunque un programma di esempio che memorizza all'interno di un file informazioni su un gruppo di alunni inserite da tastiera (Listato 13.9).

```
/* Memorizza in un file le informazioni passate dall'utente sugli alunni di un
classe */

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

/* La struttura alunno contiene nome, cognome */
/* ed età di ogni alunno                      */
struct alunno {
   char nome[100];
   char cognome[100];
   int eta;
};


main()
{
struct alunno alunno;
int nalunni;
int fd;

/* Apre il file alunni */
```

```
fd = open("alunni",O_WRONLY|O_CREAT|O_TRUNC,0600);
if( fd < 0 ) {
   printf("Non posso aprire il file alunni\n");
   exit(1);
}

printf("Quanti alunni vuoi inserire ? ");
scanf("%d",&nalunni);

while( nalunni-- > 0 ) {
   printf("Nome : ");
   scanf("%s",alunno.nome);
   printf("Cognome : ");
   scanf("%s",alunno.cognome);
   printf("Età: ");
   scanf("%d",&alunno.eta);
   write(fd, &alunno, sizeof(struct alunno));
}
close(fd);
}
```

Listato 13.9 Programma che memorizza all'interno di un file informazioni su un gruppo di alunni inserite da tastiera