Università degli Studi di Milano

Corso di Laurea in Sicurezza dei Sistemi e delle Reti Informatiche

Lezione 13 e 14 – Introduzione ai file

FABIO SCOTTI

LABORATORIO DI PROGRAMMAZIONE PER LA SICUREZZA

1. Concetti di base

1.1. Perché i file

La memorizzazione dei dati nelle variabili e nei vettori è temporanea. Non appena un programma terminerà la sua esecuzione (ovvero quando raggiungerà l'istruzione *exit*) tutti i nostri dati andranno persi. La funzione dei file è quella di permettere la memorizzazione di piccole e grandi quantità di dati in un elaboratore. I file sono memorizzati sui dispositivi di memorizzazione secondaria quali dischi e più raramente nastri. Oggi giorno si sono aggiunti i CDROM, i DVD, le pendrive USB, persino le macchine fotografiche digitali dispongono di un sistema di memorizzazione e trasmissione delle foto attraverso file.

1.2. File visti come stream

Nelle lezioni precedenti abbiamo già introdotto le operazioni di input/output limitandoci a quelle che coinvolgevano la tastiera come dispositivo di input e lo schermo come dispositivo di output. In quegli esempi stavamo gestendo flussi di dati con il concetto di **stream**.

Il linguaggio C gestisce dei file esattamente nello stesso modo. Il modo di ragionare che impiegavamo per gestire e formattare l'input e output da tastiera/video è ancora valido. E' necessario solo aggiungere qualche nuovo concetto.

2. Cosa serve sapere

1.3. Rappresentazione delle informazioni nei file

Potendo analizzare con uno strumento la polarizzazione magnetica della superficie del nostro disco fisso, ci accorgeremmo che tutti i nostri file memorizzati sul disco sono semplicemente composti da 1 e 0.

Perché allora si parla di file di caratteri e non solo di file binari? La risposta è ancora quella data nelle lezioni sulla rappresentazione dell'informazione: lo stesso numero scritto in binario sul disco può

essere letto come un numero (ad esempio un intero short senza segno) oppure un carattere (un elemento della tabella ASCII). Anche per i file dobbiamo sapere in che modo i valori binari memorizzati sui dischi dovranno essere interpretati.

1.4. Come usare i file pensandoli come stream

Apertura, accesso, chiusura

In C le operazioni di I/O vengono semplificate attraverso l'uso degli stream, i quali altro non sono che delle astrazioni rappresentative di un file o di un dispositivo fisico, che vengono manipolate attraverso l'uso di puntatori.

L'enorme vantaggio di usare gli stream è quello di potersi riferire ad un identificatore senza preoccuparsi di come questo venga implementato.

generalmente le operazioni che si compiono su uno stream sono tre:

1. lo si apre,
2. vi si accede (nel senso di operazioni di lettura e scrittura)
3. e lo si chiude..

Bufferizzazione

L'altra importante caratteristica è che lo stream è bufferizzato, ovvero viene riservato un buffer per evitare ritardi o interruzioni nella fase di lettura e scrittura.

Il contenuto del buffer non viene mandato al dispositivo (che si tratti di un dispositivo fisico o di un file non ha importanza) fino a quando non viene svuotato o chiuso.

Questo significa che ogni volta viene letto da un file o scritto su di esso un "pezzo" di dimensioni stabilite attraverso alcune aree temporanee di immagazzinamento (il buffer).
Questo metodo rende efficiente l'I/O, ma e' necessario fare attenzione: i dati scritti in un buffer non compaiono nel file (o nel device) finché il buffer non e' riempito o scaricato ("\n" serve a

questo). Qualsiasi uscita anormale del programma può causare problemi. ATTENZIONE QUINDI

Stream predefiniti in C

Gli stream predefiniti nel linguaggio C (che vengono messi a disposizione includendo la <stdio.h>) sono:

- **stdin**,
- **stdout**,
- **stderr**

Questi stream utilizzano principalmente il testo come metodo di comunicazione I/O.

I primi due possono essere usati con i file, con i programmi, con la tastiera, con la consolle e lo schermo.

Lo stderr può scrivere soltanto su console o sullo schermo.

La console è il dispositivo di default per stdout e stderr, mentre per stdin il dispositivo di default è la tastiera.

Le funzioni fflush e fclose

Infine gli stream, qualunque uso ne sia stato fatto, devono essere prima "puliti" e poi chiusi, questo si può fare comodamente con le funzioni fflush e fclose, formalizzate come segue:

```
fflush(FILE *stream);
fclose(FILE *stream);
```

Librerie necessarie

Per poter operare correttamente è necessario includere l'header file **<stdio.h>** che contiene tutte funzioni per l'input/output, comprese quelle che operano sui file

3. Funzioni per la gestione dei file.

1.5. Funzione fopen

Come menzionato prima, la prima cosa da fare è aprire un file; per fare ciò si usa la funzione fopen, strutturata nel seguente modo:

FILE *fopen(char *nome, char *modo);

che prende come parametri di input il nome del file al quale si intende accedere ed il modo in cui si vuole aprirlo, conforme al seguente schema:

"r" - lettura;

"w" – scrittura e creazione;

"a" - scrittura in fondo al file (append).

restituendo un puntatore all'oggetto FILE che servirà, dopo l'apertura, per poter accedere correttamente allo stream; se non si può accedere al file, viene restituito un puntatore a NULL. Qui di seguito proponiamo un semplice programma per poter leggere un file, ad esempio, di nome miofile.txt;

Esempio

```
#include <stdio.h>

int main()
{
   /* dichiara lo stream e il prototipo della funzione fopen */
   FILE *stream, *fopen();

   /* apre lo stream del file */
   stream = fopen("miofile.txt", "r");

   /* controlla se il file viene aperto */
   if (stream == NULL)
   {
      printf("Non posso aprire il file %s\n", "miofile.txt");
      exit(-1);
   }

   [...]
   /* Codice che lavora sul file */
```

```
   [...]

   /* Ecco la chiusura */
   fflush(stream);
   fclose(stream);
   return 0;
}
```

1.6. Funzioni fscanf e fprintf

Le funzioni che useremo per leggere e scrivere i file sono *fscanf* e *fprintf*. Il loro funzionamento è del tutto simile a quello delle funzioni scanf e printf che utilizziamo da molte lezioni, solo che al posto che leggere dalla tastiera e stampare a schermo, fscanf e fprintf leggono da file e stampano da file.

Con piacere quindi si nota che i concetti riguardanti la formattazione dell'input e dell'output che abbiamo già acquisito sono in realtà generali rispetto alla gestione di tutti gli stream.

Il prototipo delle due funzioni è il seguente:

```
int fprintf(FILE *stream1, char *formato, argomenti ...);
int fscanf(FILE *stream2, char *formato, argomenti ...);
```

La fprintf, come si può intuire, scrive sullo stream `stream1`, (un file, il terminale, un socket, ecc.), mentre la fscanf legge dallo stream `stream2` (un file, la tastiera, un socket, ecc.);

La stringa *formato* ha due tipi di argomenti, i caratteri ordinari che vengono copiati nello stream di output, e le specifiche di conversione, contraddistinte dal simbolo percentuale (%) e da un carattere, che illustriamo di seguito e che specifica il formato con il quale stampare le variabili presenti nella lista di argomenti:

Stringa di controllo	***Cosa viene stampato***
%d, %i	Intero decimale
%f	Valore in virgola mobile
%c	Un carattere
%s	Una stringa di caratteri
%o	Numero ottale
%x, %X	Numero esadecimale

%u	Intero senza segno
%f	Numero reale (float o double)
%e, %E	Formato scientifico
%%	Stampa il carattere %

Naturalmente la fprintf e la fscanf possono scrivere negli stream predefiniti, stdout - stderr e stdin rispettivamente, come mostrato dall'esempio seguente:

Esempio

```
/* stampa un messaggio di errore */
fprintf(stderr, "Impossibile continuare!\n");

/* scrive a video un messaggio */
fprintf(stdout, "Operazione completata!\n");

/* riceve da tastiera una stringa e la
* salva nella variabile "miastringa" */
fscanf(stdin, "%s", miastringa);
```

1.7. Leggere e scrivere un carattere su file

Esistono altre quattro funzioni che operano su file scrivendo un carattere per volta, esattamente come fanno la getchar e la putchar per gli stream predefiniti; queste quattro funzioni sono:

```
int getc(FILE *stream);
int fgetc(FILE *stream);
int putc(char ch, FILE *stream);
int fputc(char ch, FILE *stream);
```

NB. getc è in realtà una macro del preprocessore, mentre fgetc è una funzione di libreria, ma si comportano nello stesso modo. La stessa considerazione può essere estesa per putc e fputc.

1.8. Cancellare e rinominare un file

Per cancellare un file si usa:

```
int remove( char * nomefile)
```

Esempio

```
// se abbiamo una stringa con indirizzo nomefile
remove(nomefile);
```

Non si può eliminare un file aperto!

Per rinominare un file si usa:

```
int rename( char * vecchionome, char * nuovonome)
```

Esempio

```
// se abbiamo nella dir corrente il file f1.txt
rename(”f1.txt” , ”f1modif.txt”);
```

Non si può rinominare un file aperto!

1.9. Funzioni fseek() e rewind()

Sono due funzioni che permettono di riposizionare la testina (cursore) all'interno del file per poter effettuare un accesso casuale (diretto) all'interno del file leggendo i record (struttura elementare di un file dati).

La funzione **fseek** consente di spostare la testina di lettura su un qualunque byte del file:

```
int fseek (FILE *f, long offset,int origin)
```

Sposta la testina di `offset` byte a partire dalla posizione origin (che vale 0, 1 o 2).

Se lo spostamento ha successo ritorna 0.

Origine dello spostamento (costanti definite in `stdio.h`)

- 0 → inizio file, SEEK_SET
- 1 → posizione attuale nel file, SEEK_CUR
- 2 → fine file, SEEK_END

La funzione **rewind** definita come `void rewind(FILE *f);` permette di posizionare la testina all’inizio del file

1.10. Funzione feof()

La funzione feof consente di sapere se il puntatore al file e' posizionato alla fine del file:

int feof (FILE *f);

La funzione feof() restituisce vero se f punta alla fine del file e falso altrimenti.

1.11. Etichetta EOF

L'etichetta EOF, definita in stdio.h, viene utilizzata per rappresentare un marcatore di fine file (dipendente dal sistema).

Fisicamente il carattere corrispondente alla etichetta EOF si trova alla fine del file come ultimo carattere.

Esempi

Per stampare su monitor il contenuto di un file è utile usare una delle seguenti istruzioni:

```
while(!feof(stream)) {c = fgetc(stream); printf("%c",c);}
```

```
while((c= fgetc(stream)) != EOF) printf("%c",c);
```

```
while(fscanf(stream,"%c", &c) == 1) printf("%c",c);
```

Si ricorda che fscanf restituisce un intero che corrisponde al numero di parametri effettivamente letti. Nell'esempio la scanf dovrebbe leggere un solo parametro di tipo char. Quando il file è finito, ovvero la scanf non legge più caratteri, si esce dal ciclo while.

4. Un piccolo approfondimento (facoltativo) sull'I/O di basso livello

Tale forma di I/O e' UNBUFFERED, cioè ogni richiesta di read/write comporta un accesso diretto al disco (o device) scrivendo o leggendo uno specificato numero di bytes.

Non esistono facilitazioni di formato, poiché a questo livello si lavora con i bytes di informazione; in questa forma di I/O si usano binary (e non text) files. Invece di un puntatore a file si usa un "descrittore del file" che dà un unico numero intero per identificare ciascun file.

Per aprire un file si usa:

```
int open(char *filename, int flag, int perms)
```

che ritorna un file descriptor, oppure -1 se l'operazione fallisce.

Il flag controlla l'accesso al file ed ha i seguenti predefiniti valori definiti nel file fcntl.h: O_APPEND, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_WRONLY ecc. "perms" viene settato ottimamente a 0 per la maggior parte delle applicazioni.

Per creare un file si può usare la funzione:

```
creat(char *filename, int perms)
```

Per chiudere un file si usa:

```
int close(int handle)
```

Per leggere/scrivere uno specificato numero di bytes da/su un file immagazzinati in una locazione di memoria specificata da "buffer" si utilizzano:

```
int read(int handle, char *buffer, unsigned length)
int write(int handle, char *buffer, unsigned length)
```

Queste due funzioni ritornano il numero di byte letti/scritti o -1 se falliscono. Per specificare la lunghezza si utilizza, in genere, la funzione sizeof().

AD ESEMPIO:

```
/* legge un elenco di float da un file binario
il primo byte del file dice quanti float ci sono
nel file. Successivamente vengono elencati i float;
il nome del file e' letto dalla linea di comandi */
#include <stdio.h>
#include <fcntl.h>
float bigbuff[1000];
main(int argc,char **argv)
{
  int fd;
  int bytes_read;
  int file_length;

  if((fd=open(argv[1],O_RDONLY))==-1)
     { /* errore, file non aperto */
         exit(1);
     }
  if ((bytes_read=read(fd,&file_length,sizeof(int))==-1)
     { /* errore in lettura file */
     exit(1);
     }
     if (file_length>999)
     { /* file troppo grande */
     exit(1);
     }
     if((bytes_read=
          read(fd,bigbuff,file_length*sizeof(float)))==-1)
     { /* errore in lettura file aperto */
         exit(1);
     }
}
```