Libreria standard di I/O

Capitolo 5 -- Stevens

Libreria standard di I/O

- rientra nello standard ANSI C perché è stata implementata su molti sistemi operativi oltre che su UNIX

- le sue funzioni individuano il file su cui fare operazioni di I/O attraverso uno **stream** (flusso di dati) e non più attraverso un file descriptor

Da fd a stream

- Quando si crea o si apre un file con le funzioni di standard I/O, si dice che si associa uno **stream** al file
- Il valore restituito da tali funzioni è un puntatore ad una struttura di tipo FILE
- La struttura contiene tutte le info per trattare lo stream:
  - file descriptor usato per l'I/O
  - puntatore al buffer per lo stream
  - dimensione del buffer
  - contatore di caratteri nel buffer
  - etc...

Standard stream

- ogni processo ha 3 stream predefiniti che sono individuati attraverso i puntatori:

  stdin che punta allo **standard input**

  stdout che punta allo **standard output**

  stderr che punta allo **standard error**

- Essi si riferiscono agli stessi files che avevano come fd: STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO

Buffering

- Scopo del buffering è quello di usare il minimo numero di chiamate a read e write
- Le librerie standard automaticamente allocano il buffer chiamando *malloc*
- Le funzioni della libreria standard di I/O utilizzano 3 tipi di buffering
  - fully buffered
  - line buffered
  - unbuffered

Fully buffered

- Le operazioni di I/O avvengono effettivamente quando il buffer è pieno

- Il termine *flush* (=far scorrere) descrive la scrittura di un buffer standard di I/O, più in particolare esso significa “writing out” il contenuto di un buffer

Line buffered

- Le operazioni di I/O avvengono quando si incontra il carattere di *newline* sull'input o output
  - ...o se si riempie il buffer prima
- Usato tipicamente su stream che si riferiscono ad un terminale (standard input o output)

Unbuffered

- In questo caso la libreria standard di I/O non bufferizza i caratteri
- Le operazioni di I/O avvengono immediatamente
  - Lo stream standard error è un esempio (per visualizzare gli errori appena possibile)

Defaults

- Standard error è *unbuffered*

- Tutti gli stream sono *fully buffered*
  - ...tranne quando si riferiscono a **terminal device**, allora sono *line buffered*

se vogliamo possiamo cambiare le modalità di buffering

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

int main(void)
{
 char *stringa="Uno alla volta?";
 while (*stringa) {
    putchar(*stringa++);
    sleep(1);  /* fermiamo il processo per un secondo */
  }
 putchar('\n');
 sleep(4);
 return(0);
}
```

Modifica del *buffering*

```
#include <stdio.h>


void  setbuf (FILE *fp, char *buf );

int    setvbuf (FILE *fp, char *buf, int mode, size_t size);
```

Restituiscono: 0 se OK,

≠0 in caso di errore

Parametri

alloca un suo buffer di lunghezza specificata in *st_blksize* nella struct *stat*

definita in <stdio.h>

Function	*mode*	*buf*	Buffer & length	Type of buffering
setbuf		nonnull	user *buf* of length BUFSIZ	fully buffered or line buffered
		NULL	(no buffer)	unbuffered
setvbuf	_IOFBF	nonnull	user *buf* of length *size*	fully buffered
		NULL	system buffer of appropriate length	
	_IOLBF	nonnull	user *buf* of length *size*	line buffered
		NULL	system buffer of appropriate length	
	_IONBF	(ignored)	(no buffer)	unbuffered

Funzioni setbuf e setvbuf

- Queste funzioni devono essere chiamate:
  - Dopo che lo stream è stato aperto (per avere il puntatore al file)
  - Prima di ogni altra operazione sullo stream

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

int main(void)
{
 char *stringa="Uno alla volta?";  /*questa volta SI*/
 setbuf(stdout, NULL);  /*stdout unbuffered */
 while (*stringa) {
     putchar(*stringa++);
     sleep(1);   /*fermiamo il processo per un secondo*/
  }
      return(0);
}
```

Funzione fflush

In ogni momento possiamo forzare il *flush* di uno stream

```
#include <stdio.h>

int fflush(FILE *fp);
```

Descrizione: scrive il contenuto del buffer sul file puntato da fp

Restituisce: 0 se OK,
EOF in caso di errore

int fflush(NULL); Effettua il *flush* di tutti gli stream aperti!

```c
#include <stdio.h>

int main(void)
{
  char *stringa="Uno alla volta?";
  while (*stringa) {
    putchar(*stringa++);
    sleep(1);   /*fermiamo il processo per un secondo*/
   }
   fflush(stdout);  /*invece di putchar('\n') */
   sleep(4);
   return(0);
}
```

System Call
vs
Funzione di libreria I/O

- una system call di I/O viene invocata ed immediatamente eseguita
- l'esecuzione di una funzione di libreria di I/O passa attraverso il buffer

richiamiamo esempi già visti nelle lezioni precedenti e facciamo delle riflessioni

Aprire uno *stream*

```
#include <stdio.h>

FILE *fopen(const char *pathname, const char *type);

FILE *freopen(const char *pathname, const char *type, FILE *fp);

FILE *fdopen(int fd, const char *type);
```

Restituiscono: un puntatore a file se OK,
NULL in caso di errore

Aprire uno *stream*

FILE *fopen(const char **pathname*, const char **type*);

Descrizione: apre il file pathname

FILE *freopen(const char **pathname*, const char **type*, FILE **fp*);

Descrizione: apre il file pathname sullo stream fp, chiudendo questo se era già aperto

FILE *fdopen(int *fd*, const char **type*);

Descrizione: prende un file descriptor (che è stato ottenuto per esempio con una open) e gli associa uno standard I/O stream

Tutte devono specificare l'utilizzo che si vuole fare di tale file aperto

Campo type

type	Description
`r or rb`	open for reading
`w or wb`	truncate to 0 length or create for writing
`a or ab`	append; open for writing at end of file, or create for writing
`r+ or r+b or rb+`	open for reading and writing
`w+ or w+b or wb+`	truncate to 0 length or create for reading and writing
`a+ or a+b or ab+`	open or create for reading and writing at end of file

- La **b** dovrebbe permettere al sistema di differenziare tra file testo e file binari; in UNIX non esiste tale differenza e quindi la presenza di **b** non ha effetto
- Il significato dei tipi riferiti a fdopen è un po' differente avendo già il file descriptor fd, avendo cioè già aperto il file
  - **w** non tronca il file
  - **a** non può creare il file
- Se un nuovo file è creato specificando **w** op **a** non siamo in grado di specificarne i permessi di accesso

Funzione fclose

```
#include <stdio.h>

int fclose(FILE *fp);
```

Descrizione: chiude uno stream aperto

Restituisce: 0 se OK,

EOF in caso di errore

Funzione fclose

- Ogni dato output presente nel buffer è “flushed” prima che il file sia chiuso
- Se la libreria standard di I/O ha automaticamente allocato un buffer per quello stream, esso viene rilasciato
- Quando un processo termina
  - tutti gli stream di I/O con dati bufferizzati non scritti sono “flushed”
  - tutti gli stream di I/O aperti sono chiusi

Lettura e scrittura di uno stream

Una volta che uno stream è stato aperto possiamo scegliere :

- I/O non formattato
  - Un carattere alla volta
    - getc, getchar, putc, putchar ...
  - Una linea alla volta
    - fgets, fputs ...
  - Diretto (I/O binario, record oriented, structure oriented)
    - fread, ...
- I/O formattato
    - scanf, printf, scanf, sprintf, fscanf, fprintf

Input di carattere

- getc, fgetc, getchar restituiscono un carattere anche se in realtà il tipo di ritorno è un intero
  - Infatti, il carattere restituito è “unsigned char” (per poter coprire tutti i possibili caratteri) che è poi convertito in “int” per gestire l’errore e la fine del file che in genere vengono individuati con un negativo
- getc può essere implementata come una macro
- fgetc è una funzione e come tale richiede più tempo di getc
- getchar ≡ getc(stdin)

EOF

- Le funzioni precedenti restituiscono *EOF* sia su errore che quando incontrano la fine del file

- In molte implementazioni sono mantenuti due flag per ogni **stream**:
  - flag di errore
  - flag di end-of-file

- Per testare il flag settato da queste funzioni si ricorre alle 2 funzioni successive

EOF

```
#include <stdio.h>


int ferror (FILE * fp );

int feof (FILE * fp );
```

Both return: nonzero (true) if condition is true, 0 (false) otherwise

```
void clearerr (FILE * fp);
```

Posizionamento in uno stream

#include <stdio.h>

long ftell(FILE *fp);

Restituisce: l'indicatore della posizione corrente
(misurato in byte) se OK, -1 su errore.

long fseek(FILE *fp, long offset, int whence); /* simile a lseek */

Restituisce: 0 se OK, ≠ 0 su errore.

void rewind(FILE *fp); /* lo stream è settato all'inizio del file */

Ancora posizionamento

int fgetpos(FILE *_fp_, fpos_t *_pos_);

int fsetpos(FILE *_fp_, const fpos_t *_pos_);

Descrizione: fgetpos (fsetpos) pone nell'oggetto (prende dall'oggetto) puntato da pos l'indicatore della posizione del file fp

Restituiscono: 0 se OK, $\neq$ 0 su errore

- fgetpos utilizzata per memorizzare una posizione da riutilizzare in seguito con fsetpos per riposizionare

File descriptor

```
#include <stdio.h>

int fileno(FILE *fp);
```

Restituisce: il file descriptor associato allo stream

- utile se vogliamo chiamare per esempio la *dup*

Esercizi 4.1 e 4.2

- Risolvere l'esercizio dell'inversione di un file utilizzando gli stream e le funzioni di I/O che leggono o scrivono un carattere alla volta.

- Rendere unbuffered gli stream utilizzati realizzando una funzione my_setbuf() (che funzioni come setbuf()) implementata utilizzando la funzione setvbuf().