Laboratorio di Sistemi Operativi

Marzo-Giugno 2008
Matricole congrue 0 mod 3

Libreria standard di I/O

Libreria standard di I/O

- Permette di fare I/O senza doversi preoccupare di problematiche di bufferizzazione
  - Definizione del buffer
  - Taglia del buffer

Il funzionamento della bufferizzazione con lo standard I/O crea spesso confusione se non si sta attenti!!!!

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 è incontrato 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 op's di I/O avvengono immediatamente
  - lo stream standard error è un esempio (per dare gli errori appena possibile)

Tipica implementazione

- 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 buffer


```
#include <stdio.h>

int main(void)
{
 char *stringa="Uno alla volta?";
 while (*stringa) {
    putchar(*stringa++);
    sleep(1);  /* fermiano 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 o di lunghezza specificata in *st_blksize* nella struct *stat o di BUFSIZ*

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 operazioni devono essere chiamate
  - dopo che lo stream è stato aperto (per avere il puntatore al file)
  - prima di ogni altra operazione sullo stream


```
#include <stdio.h>

int main(void)
{
 char *stringa="Uno alla volta?";  /*questa volta SI*/
 setbuf(stdout, NULL);  /*stdout unbuffered */
 while (*stringa) {
     putchar(*stringa++);
     sleep(1);    /*fermiano 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 fa fp

Restituisce: 0 se OK,
EOF in caso di errore


```
#include <stdio.h>

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

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

tipi

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, ...
  - Una linea alla volta
    - fgets, ...
  - Diretto (I/O binario, record oriented, structure oriented)
    - fread, ...
- I/O formattato
    - fscanf, fprintf ...

input di un carattere

```
#include <stdio.h>

int getc(FILE *fp);
int fgetc(FILE *fp);
int getchar(void);
```

Descrizione: leggono il prossimo carattere dal file puntato da *fp*

Restituiscono: il prossimo carattere se OK, EOF se alla fine del file o in caso di errore

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)
  - convertito in "int" per gestire l'errore e la fine del file che in genere vengono individuati con un negativo -
    - ferror(FILE*), feof(FILE*), clearerr(FILE*)
- 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

funzioni ferror e feof

```
#include <stdio.h>

int ferror(FILE *fp);
int feof(FILE *fp);
```

Restituiscono: true se la condizione è vera (flag = 1) ,
false altrimenti

per resettare entrambi i flag c'è la funzione:
void clearerr(FILE *fp);

funzione ungetc

```
#include <stdio.h>

int ungetc(int c, FILE *fp);
```

Descrizione: mette il carattere *c* nel file puntato da *fp*, nella posizione dell'ultimo carattere richiamato da getc

Restituisce: *c* se OK,
EOF in caso di errore

funzione ungetc

- si può reinserire un carattere differente da quello letto da getc
- non si può fare ungetc di EOF
- se leggiamo (con getc) un EOF e poi immettiamo un carattere con ungetc il nuovo carattere viene immesso prima di EOF; questo significa che chiamate due getc avremo il carattere appena immesso e poi EOF

ouput di un carattere

#include <stdio.h>

int putc(int *c*, FILE **fp*);
int fputc(int *c*, FILE **fp*);
int putchar(int *c*);

Descrizione: immettono il carattere *c* nel file puntato da *fp*

Restituiscono: *c* se OK,
EOF in caso di errore

output di carattere

- putc può essere implementata come una macro
- fputc è una funzione e come tale richiede più tempo di putc
- putchar(c) ≡ putc(c, stdout)

input di una linea

#include <stdio.h>

char *fgets(char **buf*, int *size*, FILE **fp*);

char *gets(char **buf*);

Descrizione: inseriscono in *buf* la linea presa dal file che in fgets è individuato da *fp* ed in gets è lo *stdin*

Restituiscono: *buf* se OK,
NULL se alla fine del file o in caso di errore

input di linea

- fgets legge al più *size*-1 caratteri compreso anche lo \n
  - Nel buffer è sempre inserito un "null byte"
  - Se *size*-1 è minore della lunghezza della linea, la prossima fgets leggerà il resto della linea
- L' uso di gets non è consigliabile perché non contiene la size del buffer
  - Tutta la linea è letta anche se il buffer va in overflow e si finisce con scrivere su una zona di memoria che non è del buffer

ouput di una linea

#include <stdio.h>

int fputs(const char *str, FILE *fp);

int puts(const char *str);

Descrizione: scrivono il contenuto di *str* sul file che in fputs è individuato da *fp* ed in puts è *stdout*

Restituiscono: un valore non negativo se OK, EOF in caso di errore

output di linea

- fputs e puts non scrivono sul file il "null byte" contenuto in *str*
- puts aggiunge alla fine del contenuto di *str* la \n sullo standard output
  - Come per gets il suo uso non è consigliabile

I/O diretto (o anche binario)

- Con tale forma di I/O possiamo voler leggere o scrivere una intera struttura ad ogni passo
- Fare questo
  - con getc e putc sarebbe troppo oneroso dovendo prendere 1 byte alla volta
  - con fgets ed fputs non è il caso perché: fputs si blocca nella lettura quando incontra un null che potrebbe far parte della struttura ed fgets potrebbe comportarsi in maniera scorretta se incontra null op \n

I/O diretto (o anche binario)

#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nobj, FILE *fp);
size_t fwrite(const void *ptr, size_t size, size_t nobj, FILE *fp);

Restituiscono: il numero di oggetti letti o scritti

I/O diretto

- fread e fwrite sono per esempio usati per leggere o scrivere strutture
- fread e fwrite possono non funzionare bene perché sistemi differenti o opzioni di compilazione sul medesimo sistema possono essere tali che il binary layout di una struttura sia letto e scritto in maniera differente

I/O diretto: esempi

```
float data[10];
if (fwrite(&data[2], sizeof(float), 4, fp) !=4)
printf("fwrite error");
```

```
struct{
short count;
long total;
char name[NAMESIZE];
}item;
if (fwrite(&item, sizeof(item), 1, fp) !=1)
printf("fwrite error");
```


input formattato

#include <stdio.h>

int scanf(const char **format*, …);

int fscanf(FILE **fp*, const char **format*, …);

int sscanf(const char **buf*, const char **format*, …);

Descrizione: leggono rispettivamente da *stdin*, *fp* e dall'array *buf*

Restituiscono: il numero di oggetti assegnati se OK, EOF se alla fine del file o in caso di error

ouput formattato

#include <stdio.h>

int printf(const char **format*, …);

int fprintf(FILE **fp*, const char **format*, …);

int sprintf(char **buf*, const char **format*, …);

Descrizione: scrivono rispettivamente su *stdout*, *fp* e sull'array *buf*

posizionamento in uno stream

```
#include <stdio.h>
off_t ftello(FILE *fp);
```

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

```
off_t fseeko(FILE *fp, long offset, int whence); /* simile a lseek */
```

Restituisce: 0 se OK,
≠ 0 su errore.

```
void rewind(FILE *fp); /* lo stream è settao 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,
≠ 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*

Temporary files

```
#include <stdio.h>

char *tmpnam(char *ptr);
```

un pathname non nel file system

```
FILE *tmpfile(void);
```

crea un file temporaneo
il file viene eliminato quando il processo termina

esercizio

- Implementare una funzione che crea un file temporaneo usando solo la funzione tmpnam ma senza usare la funzione tmpfile()

  Ricorda che il file deve scomparire alla terminazione del processo