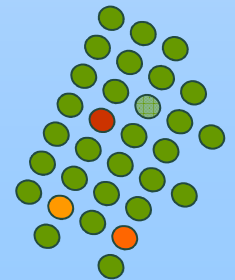


---

# Libreria standard di I/O

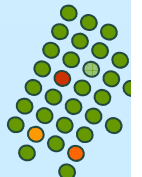
---

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 il 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:
  1. *stdin* che punta allo **standard input**
  2. *stdout* che punta allo **standard output**
  3. *stderr* che punta allo **standard error**
  
- Essi si riferiscono agli stessi files che avevano come fd: `STDIN_FILENO`, `STDOUT_FILENO`, `STDERR_FILENO`



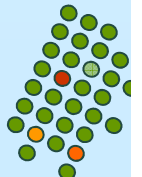
# buffering

- 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
  1. fully buffered
  2. line buffered
  3. 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



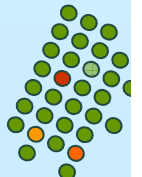
# line buffered

- 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 bafferizza i caratteri
- le op's di I/O avvengono immediatamente
  - lo stream standard error è un esempio (per dare 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 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*

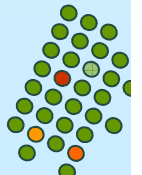
```
#include <stdio.h>
```

```
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

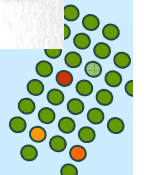


# parametri

alloca un suo buffer di lunghezza specificata in *st\_blksize* nella struct *stat*

definita in `<stdio.h>`

Function	<i>mode</i>	<i>buf</i>	Buffer & length	Type of buffering
setbuf		nonnull	user <i>buf</i> of length BUFSIZ	fully buffered or line buffered
		NULL	(no buffer)	unbuffered
setvbuf	_IOFBF	nonnull	user <i>buf</i> of length <i>size</i>	fully buffered
		NULL	system buffer of appropriate length	
	_IOLBF	nonnull	user <i>buf</i> of length <i>size</i>	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);
```

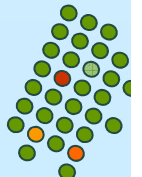
Descrizione: scrive il contenuto del buffer sul file puntato da fp

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*

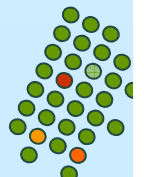
```
#include <stdio.h>
```

```
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

<i>type</i>	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** o **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 ...`



# 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`  $\equiv$  `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





# 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.



# 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



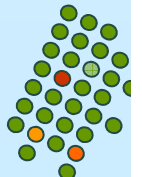
# file descriptor

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

```
int fileno(FILE *fp);
```

Restituisce: il file descriptor associato allo stream

- utile se vogliamo chiamare per esempio la *dup*



## esercizio 4.1

- 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()`.

