


(Stevens)

Capitolo 3: File I/O





I/O di base

- open
- read
- write
- creat
- close
- lseek





File descriptor

- Un file descriptor (***fd***) e' un intero non negativo associato ad un file.
- Per il kernel tutti i file aperti sono identificati tramite file descriptor.
- Le funzioni di I/O identificano i file per mezzo dei file descriptor.
- Quando un file viene aperto o creato il kernel restituisce al processo che lo ha aperto o creato un file descriptor ***fd***.
 - ◆ da questo momento per ogni operazione sul file verrà usato ***fd***.
- Un file descriptor sarà un intero compreso tra:
 $0 \leq fd \leq \text{OPEN_MAX}$
 - ◆ La costante OPEN_MAX e' definita in <limits.h>





Standard files

- Ogni nuovo processo apre 3 file standard
 - ◆ input
 - ◆ output
 - ◆ error
- e fa riferimento ad essi tramite i tre file descriptor
 - ◆ 0 (STDIN_FILENO)
 - ◆ 1 (STDOUT_FILENO)
 - ◆ 2 (STDERR_FILENO)





open

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

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

```
#include <fcntl.h>
```

```
int open(const char *pathname, int oflag, ... /* , mode_t mode */);
```

Restituisce: un file descriptor se OK

-1 altrimenti

- Il file descriptor restituito è il più piccolo intero positivo non ancora usato come file descriptor.





open (II)

- L'argomento ***oflag*** indica una moltitudine di opzioni.
- E' formato dall'OR di una o più delle seguenti costanti simboliche definite in <fcntl.h>
 - ✦ Una sola costante tra O_RDONLY (aperto per sola lettura), O_WRONLY (aperto per sola scrittura), O_RDWR (aperto per lettura e scrittura)
 - ✦ Una qualunque tra (sono opzionali)
 - ✓ O_APPEND = tutte le write avverranno alla fine del file
 - ✓ O_CREAT = se il file non esiste viene creato (richiede il terzo argomento ***mode***).
 - ✓ O_EXCL = se O_CREAT e' specificato ed il file esiste genera un errore.
 - ✓ O_TRUNC = se il file già esiste, tronca la sua lunghezza a 0
 - ✓ (ed altre) ...





open (mode)

- L'argomento **mode** viene utilizzato quando si crea un nuovo file utilizzando **O_CREAT** per specificare i permessi di accesso del nuovo file che si sta creando.
- Se il file già esiste questo argomento è ignorato.

Costanti per il mode

mode	Description	
S_ISUID	set-user-ID on execution	4000
S_ISGID	set-group-ID on execution	2000
S_ISVTX	saved-text (sticky bit)	1000
S_IRWXU	read, write, and execute by user (owner)	700
S_IRUSR	read by user (owner)	400
S_IWUSR	write by user (owner)	200
S_IXUSR	execute by user (owner)	100
S_IRWXG	read, write, and execute by group	070
S_IRGRP	read by group	040
S_IWGRP	write by group	020
S_IXGRP	execute by group	010
S_IRWXO	read, write, and execute by other (world)	007
S_IROTH	read by other (world)	004
S_IWOTH	write by other (world)	002
S_IXOTH	execute by other (world)	001





Controllo degli accessi: Unix

- Modi di accesso: lettura, scrittura, esecuzione
- Tre classi di utenti:

			RWX
a) accesso proprietario	7	⇒	1 1 1
			RWX
b) accesso gruppo	6	⇒	1 1 0
			RWX
c) accesso pubblico	1	⇒	0 0 1





read

#include <unistd.h>

ssize_t read (int filedes, void *buff, size_t nbytes);

- Descrizione: legge dal file con file descriptor ***filedes*** un numero di bytes che e' al più ***nbyte*** e li mette in ***buff***.

Restituisce: il numero di bytes letti,
 0 se alla fine del file
 -1 (errore) altrimenti





read (II)

- La lettura parte dal current offset
- Alla fine il current offset è incrementato del numero di byte letti
- Se nbytes=0 viene restituito 0 e non vi è altro effetto
- Se il current offset è alla fine del file o anche dopo, viene restituito 0 e non vi è alcuna lettura
- Se c'è un "buco" (ci sono byte in cui non è stato scritto) nel file, vengono letti byte con valore 0





write

#include <unistd.h>

ssize_t write(int filedes, const void *buff, size_t nbytes);

■ Descrizione: scrive ***nbyte*** presi dal ***buff*** sul file con file descriptor ***filedes***

Restituisce: il numero di bytes scritti se OK
 -1 altrimenti





write (II)

- La posizione da cui si comincia a scrivere è il current offset
- Alla fine della scrittura, current offset viene incrementato di **nbytes**
- Se la scrittura ha causato un aumento della lunghezza del file anche questo attributo viene aggiornato
- Se viene richiesto di scrivere più byte rispetto allo spazio a disposizione (es: limite fisico di un dispositivo di output), solo lo spazio disponibile è occupato e viene restituito il numero effettivo di byte scritti (**$\leq \text{nbytes}$**)
- Se **filedes** è stato aperto con O_APPEND allora current offset è settato alla fine del file in ogni operazione di **write**
- Se **nbytes=0** viene restituito 0 e non vi è alcuna scrittura





creat

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

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

```
#include <fcntl.h>
```

```
int creat (const char *pathname, mode_t mode );
```

■ Descrizione: crea un file dal nome **pathname** con i permessi descritti in **mode**

Restituisce: file descriptor del file **aperto come write-only** se OK,
-1 altrimenti

E' equivalente a:

```
open( pathname, O_WRONLY | O_CREAT | O_TRUNC , mode),
```





close

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

```
int close( int filedes );
```

- Descrizione: chiude il file con file descriptor **filedes**

Restituisce: 0 se OK
-1 altrimenti

- Quando un processo termina, tutti i file aperti vengono chiusi automaticamente dal kernel
- Quindi molti programmi non chiudono esplicitamente i file aperti perchè saranno automaticamente chiusi alla terminazione del processo.





offset

- Ogni file aperto ha assegnato un current offset (intero > 0) che misura in numero di byte la posizione raggiunta nel file
- Operazioni come **open** e **creat** settano il current offset all'inizio del file se O_APPEND non è specificato
- Operazioni come **read** e **write** partono dal current offset e lo incrementano del numero di byte letti o scritti





lseek

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

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

```
off_t lseek (int filedes, off_t offset, int whence );
```

Restituisce: il nuovo offset se OK
-1 altrimenti





lseek (whence)

- L'argomento **whence** può assumere valore
 - ◆ SEEK_SET
 - ✓ ci si sposta del valore di offset a partire dall'inizio
 - ◆ SEEK_CUR
 - ✓ ci si sposta del valore di offset (positivo o negativo) a partire dalla posizione corrente
 - ◆ SEEK_END
 - ✓ ci si sposta del valore di offset (positivo o negativo) a partire dalla fine del file





lseek (II)

- **lseek** permette di settare il current offset oltre la fine dei dati esistenti nel file.
- Se vengono inseriti successivamente dei dati in tale posizione, una lettura nel buco restituirà byte con valore 0
- In ogni caso **lseek** non aumenta la taglia del file
- Se **lseek** fallisce (e restituisce -1) il valore del current offset rimane inalterato





Esercizio

- Usando le funzioni C:

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

```
#include <ctype.h>
```

```
int islower(char c);
```

```
int isupper(char c);
```

```
int tolower(char c);
```

```
int toupper(char c);
```

- ed anche:

open, read, write, close

- Scrivere un programma C che prenda come input da linea di comando due nomi di file F1 ed F2 e scriva in F2 il contenuto di F1 invertendo maiuscole e minuscole.

