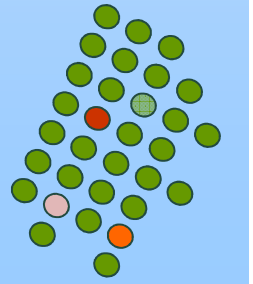
I/O non bufferizzato (1)

Capitolo 3 -- Stevens



System Call

- open
- close
- read
- write
- lseek

file descriptor

- sono degli interi non negativi
- il kernel assegna un file descriptor ad ogni file aperto
- le funzioni di I/O identificano i file per mezzo dei fd
  - nota la differenza con ANSI C
    - fopen, fclose → FILE *file_pointer

file descriptor...ancora

- per riferirsi ai file si comunica con il kernel tramite i file descriptor
- all'apertura/creazione di un file, il kernel restituisce un fd al processo
- da questo momento per ogni operazione su file usiamo il fd per riferirci ad esso
- 0 <= fd < = OPEN_MAX
  - …limiti di OPEN_MAX
    - 19 in vecchie versioni di UNIX
    - 63 in versioni più recenti
    - Limitato dalla quantità di memoria nel sistema (SVR4)

standard file

- Ogni nuovo processo apre 3 file standard
  - input
  - output
  - error
- e vi si riferisce con 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>
```

In linux è <include/fcntl.h>

int open(const char *_pathname_, int _oflag_, … /* , mode_t _mode_ */);

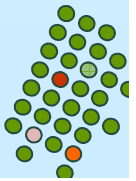
Restituisce: un fd se OK

-1 altrimenti

- fd restituito è il più piccolo numero non usato come fd

open

- L'argomento *oflag* è formato dall'OR di uno o più dei seguenti flag di stato (<include/fcntl.h>)
  - Una ed una sola costante tra
    - O_RDONLY, O_WRONLY, O_RDWR
  - Una qualunque tra (sono opzionali)
    - O_APPEND = tutto ciò che verrà scritto sarà posto alla fine
    - O_CREAT = usato quando si usa open per creare un file
    - O_EXCL = messo in Or con O_CREAT per segnalare errore se il file già esiste
    - O_TRUNC = se il file già esiste, aperto in write op read-write tronca la sua lunghezza a 0
    - O_SYNC (SVR4) = se si sta aprendo in write, fa completare prima I/O
    - O_NOCTTY, O_NONBLOCK



open

- 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

```c
#include  <sys/types.h>
#include  <fcntl.h>
#include  <sys/stat.h>

int main(void)
{
  int    fd;

   fd=open(“FILE",O_RDONLY);
   exit(0);
}
```

```c
#include  <sys/types.h>
#include  <fcntl.h>
#include  <sys/stat.h>

int main(void)
{
  int    fd;
   fd=open(“FILE1",O_CREAT|O_EXCL|O_WRONLY,0700);
   exit(0);
}
```


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: fd del file aperto come write-only se OK,
-1 altrimenti

open(*pathname*, O_WRONLY | O_CREAT | O_TRUNC, *mode*);

open(*pathname*, O_RDWR | O_CREAT | O_TRUNC, *mode*);

umask

#include <sys/types.h>

#include <sys/stat.h>

mode_t umask (mode_t *cmask*);

Descrizione: setta la maschera di creazione per

l’accesso ad un file

Restituisce: la maschera di creazione precedente

(nota che non restituisce valori di errori)

umask

- Viene usato ogni volta che il processo crea un nuovo file o directory, secondo il seguente criterio:
  - setta la maschera di creazione (*cmask*)
  - comando: umask
  - alla craezione del file viene fatto l'AND tra la maschera negata e il *mode* della creazione del file

umask: 022 (--- -w- -w-)	000 010 010	NOT
negaz.: 755 (rwx r-x r-x)	111 101 101	AND
creat("pippo.txt", 665)	110 110 101	
rw- r– r-x	110 100 101	

?

- qual'è la maschera di default di creazione di file?

```
#include     <sys/types.h>
#include     <sys/stat.h>
#include     <fcntl.h>
#include     "ourhdr.h"  /*esempio di utilizzo di umask*/
int main(void)
{
  umask(0);
  if(creat("foo",S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|
                                        S_IROTH|S_IWOTH)<0)
      err_sys("creat error for foo");

   umask(S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);/* 0066 */
   if (creat("bar",S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP|
                                    S_IROTH|S_IWOTH)<0)
      err_sys("creat error for bar");
   exit(0);
}
```


Risultati programma precedente

- rw- rw- rw- foo
- rw- --- --- bar

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 automaticamente chiusi dal kernel

offset

- ogni file aperto ha assegnato un current offset (intero >0) che misura in numero di byte la posizione nel file
- Operazioni come open e creat settano il current offset all'inizio del file ammeno che O_APPEND è specificato (open)
- operazioni come read e write partono dal current offset e causano un incremento pari al numero di byte letti o scritti

lseek

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

#include <unistd.h>


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

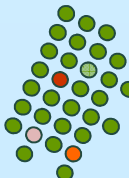
Restituisce: il nuovo offset se OK

-1 altrimenti

lseek

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 (taglia) del file



lseek

- 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 (restituisce –1), il valore del current offset rimane inalterato

```
#include  <sys/types.h>
#include  <fcntl.h>
#include  <sys/stat.h>
#include  <unistd.h>

int main(void)
{
  int    fd,i;

   fd=open(“FILE",O_RDONLY);
   i=lseek(fd,50,SEEK_CUR);
   exit(0);
}
```


read

#include <unistd.h>

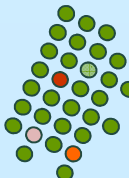
ssize_t read (int *filedes*, void **buff*, size_t *nbytes*);

Descrizione: legge dal file con file descriptor *filedes* un numero di byte *nbyte* e li mette in *buff*

Restituisce: il numero di bytes letti,

0 se alla fine del file

-1 altrimenti



read

- 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

```
#include  <sys/types.h>
#include  <fcntl.h>
#include  <sys/stat.h>
#include  <unistd.h>

int main(void)
{
  int    fd,i;
  char   *buf;

   fd=open(“FILE",O_RDONLY);
   i=lseek(fd,50,SEEK_CUR);
   read(fd,buf,20);
   exit(0);
}
```


write

```
#include <unistd.h>

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

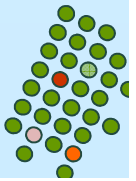
Descrizione: scrive *nbyte* presi dal *buff* sul file con file descriptor *filedes*

Restituisce: il numero di bytes scritti se OK

-1 altrimenti

write

- La posizione da cui si comincia a scrivere è current offset
- Alla fine della scrittura current offset è incrementato di *nbytes* e se tale scrittura ha causato un aumento della lunghezza del file anche tale parametro 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 (<=*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



```
#include  <sys/types.h>
#include  <fcntl.h>
#include  <sys/stat.h>
#include  <unistd.h>

int main(void)
{
  int    fd,i;
  char   *buf;

   fd=open(“FILE",O_RDONLY);
   i=lseek(fd,50,SEEK_CUR);
   read(fd,buf,20);
   write(1,buf,20);
   exit(0);
}
```


```
/* File: seek.c */
#include     <sys/types.h>
#include     <fcntl.h>
#include     "ourhdr.h"

int main(void)
{
  off_t  i;
  int    fd;
  char   *s;

   fd=open("seek.c",O_RDONLY);
   i=lseek(fd, 30, SEEK_CUR);
   printf("posizione corrente  %d\n", i);

   s=(char *) malloc(25*sizeof(char));
    read (fd, s, 20);
    printf ("leggo da: \n %s\n", s);
   exit(0);
}
```


esercizio

- programma 3.2: buco
  - far stampare il contenuto del file col buco
    - od –c
    - cat

esercizi

- Scrivere un programma in C che prende un input da tastiera e lo scrive nel file FILE1.

- Copiare in ordine inverso il contenuto di FILE1 in un file FILE2 e stampare il contenuto di FILE2 sul terminale.