Laboratorio di Sistemi Operativi

II Semestre - Marzo/Giugno 2008
matricole congr. 0 mod 3

I/O non bufferizzato

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
    - 20 in vecchie versioni di UNIX
    - 64 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
  - Zero o piu' tra le seguenti (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 o 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.

Pathname: Troncamento

- Se si cerca di creare un file con un nome troppo lungo (vedi: NAME_MAX)
  - Il sistema potrebbe troncare il nome al massimo numero di byte disponibili
  - O ritornare un errore e settare *errno* a ENAMETOOLONG
- L'opzione NO_TRUNC fornisce indicazioni sul comportamento del sistema
  - Valore con
    - pathconf(char* pathname, _PC_NO_TRUNC)

    -1 indica che la funzione non e' supportata

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*);

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 in cui avverra' la prossima operazione di scrittura o lettura
- Operazioni come open e creat settano il current offset all'inizio del file a meno 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

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

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


```
/* File: seek.c */
#include     <sys/types.h> /* per off_t() */
#include     <fcntl.h> /* per open() */
#include     <unistd.h> /* per lseek() */
/* necessita #include di stdio.h and stdlib.h */
int main(void)
{
  off_t  i;
  int    fd;
  char   *s;

   fd=open("seek.c",O_RDONLY);
   i=lseek(fd, 9, SEEK_CUR);
   printf("posizione corrente  %d\n", (int)i);

   s=(char *) malloc(25*sizeof(char));
    read (fd, s, 6);
    printf ("leggo da: \n %s\n", s);
   exit(0);
}
```

- Vediamo ora come viene creato un file con un "buco" (lseek oltre la fine-file e poi write) -> vedi file: hole.c
  - Per far stampare il contenuto del file col buco
    - od –c
    - cat

Efficienza di I/O

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#define BUFFSIZE    8192
int main(void)
{
   int   n;
   char bufBUFFSIZE];

   while((n = read(STDIN_FILENO,buf,BUFFSIZE))>0)
        if (write(STDOUT_FILENO,buf, n) != n)
              printf("\n ERROR: write error\n");
  if (n < 0)
        printf("\n ERROR: read error\n");
   exit(0);
}
```

-apertura file standard
-chiusura file

- scelta di BUFFERSIZE