Laboratorio di Sistemi Operativi

primavera 2009

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
    - 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


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

int main(void)
{
  int    fd;

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

```
#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,700);
   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);*

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

```c
#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


```c
#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

Efficienza di I/O

```
#include "ourhdr.h"

#define BUFFSIZE    8192

int main(void)
{
   int  n;
   char buf[BUFFSIZE];

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

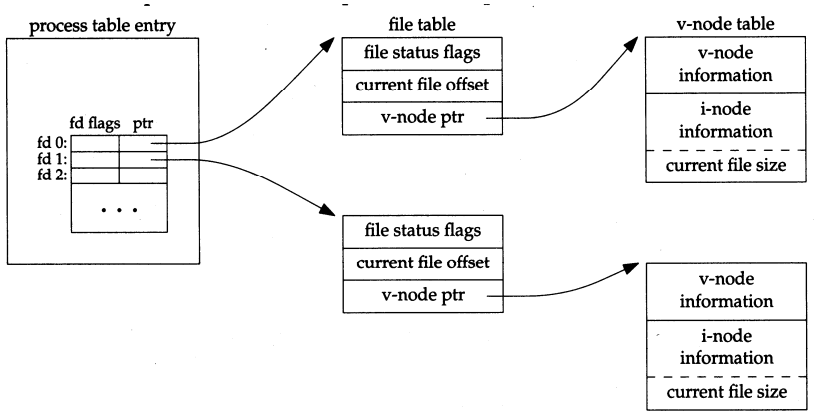
-apertura file standard
-chiusura file

- scelta di BUFFERSIZE

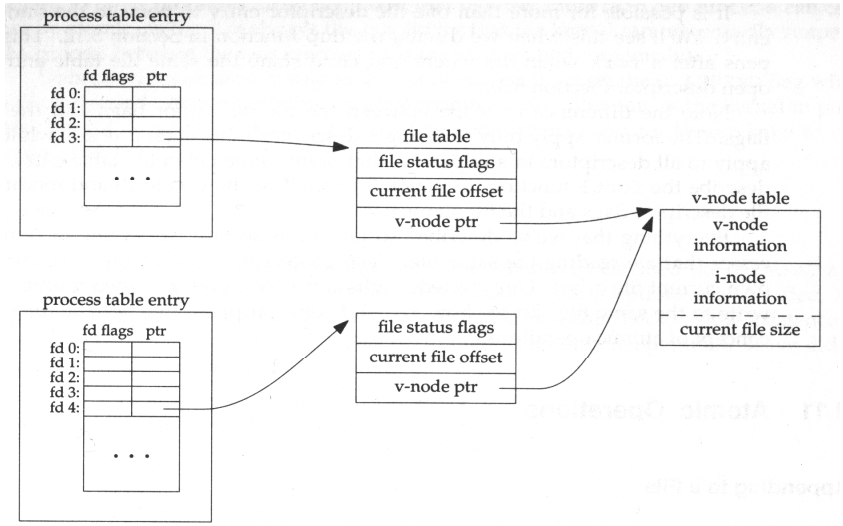
Condivisione di file

- Unix supporta la possibilità che più processi condividano file aperti
- Prima di analizzare questa situazione esaminiamo le strutture dati che il kernel utilizza per I/O
  - 3 strutture dati per l'I/O

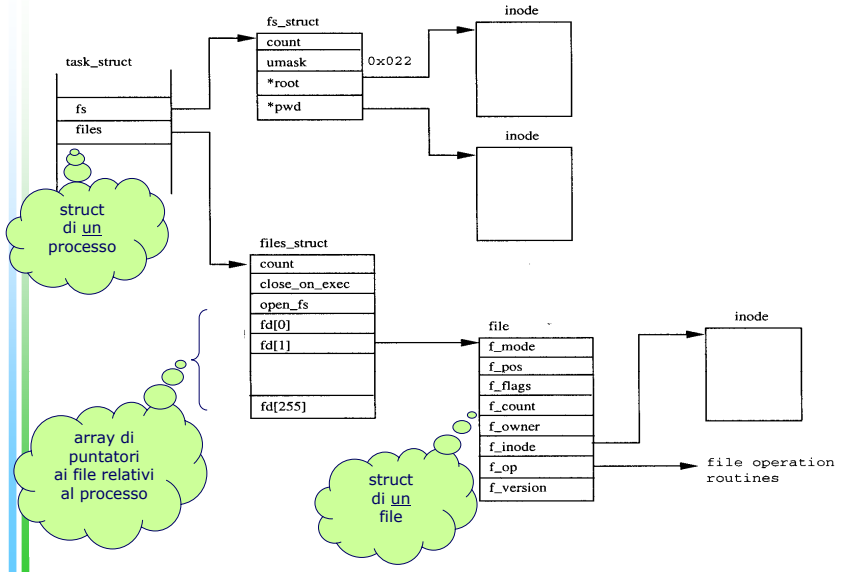
Strutture dati di file aperti



2 processi su uno stesso file



In Linux...



Laboratorio di Sistemi Operativi

Operazioni Atomiche

- Immaginate il seguente scenario:
  - 2 processi aprono lo stesso file
  - ognuno si posiziona alla fine e scrive (in 2 passi)
    1. lseek(fd, 0 ,SEEK_END);
    2. write (fd, buff , 100);
  - se il kernel alterna le due operazioni di ogni processo si hanno

    effetti indesiderati

Laboratorio di Sistemi Operativi

Operazioni Atomiche

- Unix risolve il problema:
  - Apre il file con il flag "O_APPEND"
  - Questo fa posizionare l'offset alla fine, prima di ogni write
  - In altre parole, le operazioni di
    1. posizionamento
    2. write

    } sono atomiche

- In generale una operazione atomica è composta da molti passi che o sono eseguiti tutti insieme o non ne è eseguito nessuno

Laboratorio di Sistemi Operativi

dup & dup2

#include <unistd.h>

int dup(int *filedes*);

int dup2(int *filedes*, int *filedes2*);

Descrizione: assegnano un altro fd ad un file che già ne possedeva uno, cioè *filedes*

Restituiscono entrambe: il nuovo fd se OK
-1 altrimenti

dup & dup2

In particolare:

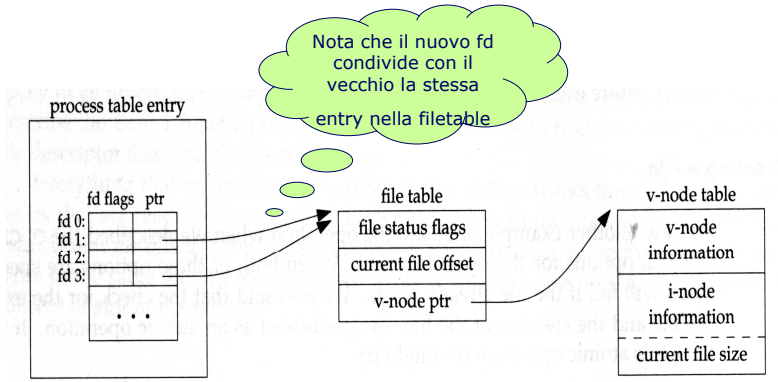
int dup(int *filedes*);

- restituisce il più piccolo fd disponibile

int dup2(int *filedes*, int *filedes2*);

- Assegna al file avente già file descriptor *filedes* anche il file descriptor *filedes2*
  - Se *filedes2* è già open esso è prima chiuso e poi è assegnato a *filedes*
  - Se *filedes2*=*filedes* viene restituito direttamente *filedes2*
- dup2 è una operazione atomica

DS del kernel, dopo dup



esercizi

1. copiare un file in un altro usando solo le funzioni di standard I/O *getchar* e *putchar*
   - hint: "duplicare" gli standard file
2. copiare il contenuto di un file in un altro usando esclusivamente read da standard input e write su standard output

fcntl

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

int fcntl (int filedes, int cmd, ... /* int arg */  );
```

Descrizione: cambia le proprietà di un file già aperto

Restituisce: un valore che dipende da *cmd* se OK
–1 altrimenti

- negli esempi successivi il terzo argomento è sempre 0

uso di fcntl

1. duplica un descrittore esistente (*cmd* = F_DUPFD)
2. prende/setta i flag del fd (*cmd* = F_GETFD o F_SETFD)
3. prende/setta i flag di stato dei file (*cmd* = F_GETFL o SETFL)
4. prende/setta proprietà di I/O asincrono (*cmd* = F_GETOWN o F_SETOWN)
5. prende/setta record lock (*cmd* = F_GETLK, F_SETLK o F_SETLKW)
   per il momento lo tralasciamo

fcntl(*filedes*, F_DUPFD,0)

- duplica il file descriptor *filedes*
  - simile a dup(*filedes*)
  - Restituisce il più piccolo descrittore che non è già aperto e che sia ≥ 3° argomento (=0)
  - vedi man 2 o Stevens per dettagli

fcntl(*filedes*, F_GETFD,0)

- restituisce i flag del file descriptor *filedes*

per il momento ne consideriamo uno solo
FD_CLOEXEC
che se è settato lascia aperto il file descriptor durante una exec

tralasciamo per il momento F_SETFD...dettagli sul libro

fcntl(*filedes*, F_GETFL,0)

- restituisce i flag di stato per *filedes* in un byte!
  - quindi per testare i flag per l'accesso dobbiamo usare la maschera *00000011* (O_ACCMODE in <fcntl.h>) con un AND bit a bit

...vediamo un esempio?


```
#include    <sys/types.h>     /*frammento estratto dal programma 3.4 */
#include    <fcntl.h>
#include    "ourhdr.h"

int main(int argc, char *argv[])
{
   int  accmode, val;
                /* prende i flag di stato del file relativo al file
                   descriptor dato in input */
   if ( (val = fcntl(atoi(argv[1]), F_GETFL, 0)) < 0)
                err_sys("fcntl error for fd %d", atoi(argv[1]));

   accmode = val & O_ACCMODE;
   if (accmode == O_RDONLY)  printf("read only");
   else if (accmode == O_WRONLY)  printf("write only");
              ..................
```


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.