Laboratorio di Sistemi Operativi

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

I/O non bufferizzato-2

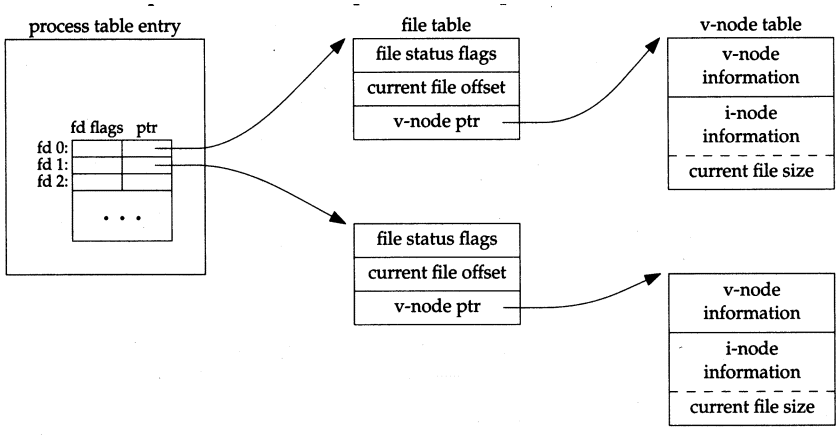
System Call

- dup, dup2
- sync, fsync, fdatasync
- fcntl

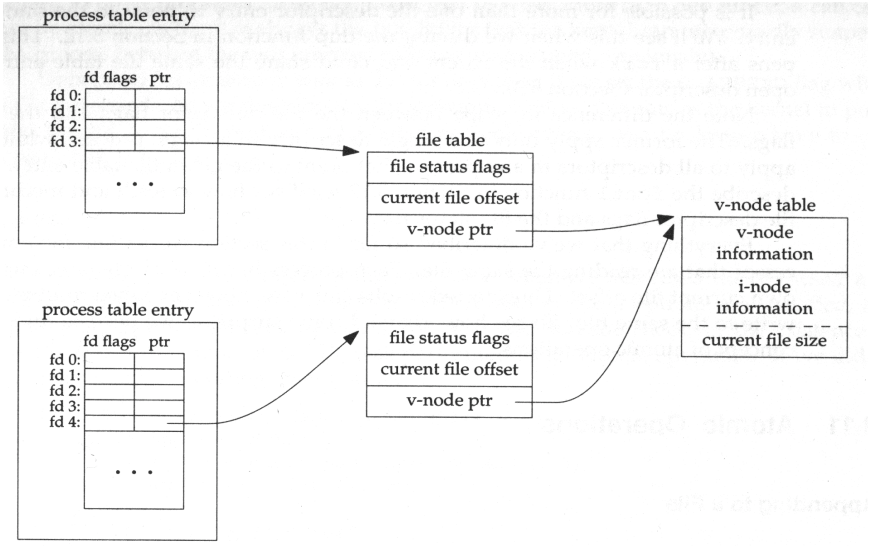
Condivisione di file

- 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

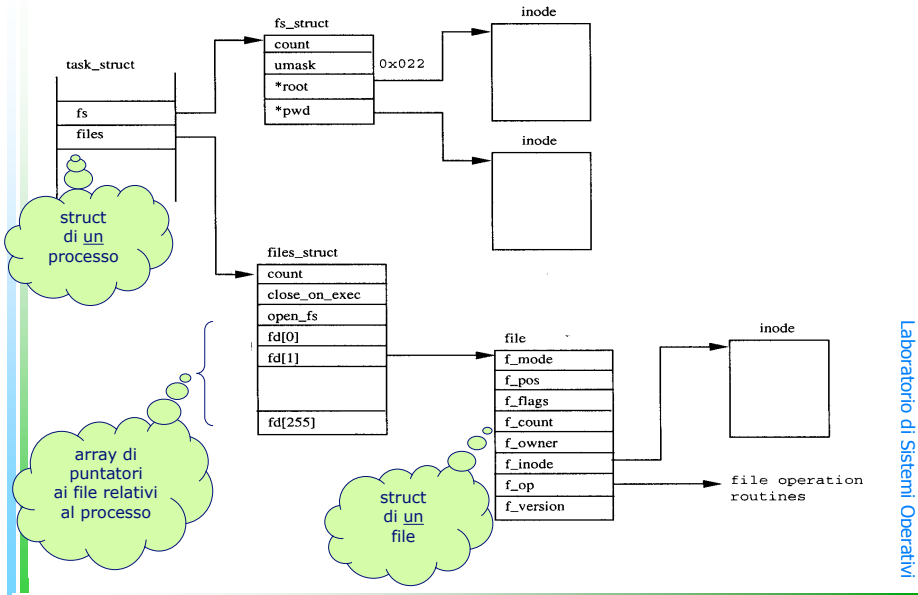
Strutture dati di file aperti



2 processi su uno stesso file



In Linux...



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

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 un' operazione atomica è composta da molti passi che o sono eseguiti tutti insieme o non ne è eseguito nessuno

pread & pwrite

```
#include <unistd.h>
ssize_t pread(int filedes, void *buf, size_t nbytes, off_t offset );
                    ritorna #byte letti, 0 se EOF raggiunto, -1 su errore

ssize_t pwrite(int filedes, const void *buf, size_t nbytes, off_t offset );
                    ritorna #byte letti se OK, -1 su errore
```

pread/pwrite equivale ad eseguire in maniera atomica

```
lseek(filedes, offset, SEEK_SET);
read/write(filedes, buf, nbytes);
```

Ma l'offset corrente del file non viene modificato

Operazioni Atomiche:Creazione

- Creare un file solo se non esiste già:

SOL1:

```
if((fd=open(fname, O_WRONLY)) <0){
        if(errno == ENOENT){
                if((fd = creat(fname, mode))<0)
                        printf("create_error");
        }else{
                printf("open_error");
        }
}
```

SOL2:

```
if((fd=open(fname,O_WRONLY|O_CREAT|O_EXCL,mode))<0)
 . . .  /* controlla il tipo di errore */
```

errori

#include <errno.h> definisce il simbolo *errno* e le costanti associate
vedere anche *man 2 intro*

#include <string.h>
char *strerror(int *errnum*);

Descrizione: trasforma errnum (tipicamente errno) in una stringa che spiega il tipo di errore

#include <stdio.h>
void perror(const char **msg*);

Descrizione: stampa la stringa *msg* seguita da ": " e quindi dal messaggio corrispondente al valore di errno

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:

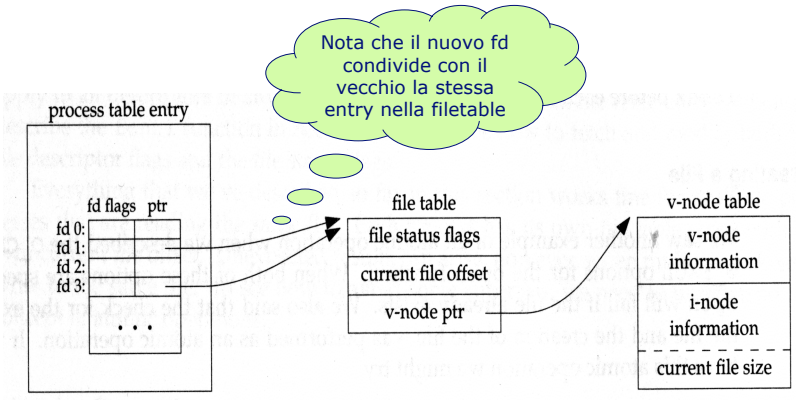
int dup(int *filedes*);

- 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



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

fcntl(*filedes*, F_SETFD,*value*)

- Setta i flag del file descriptor *filedes* al valore fornito nel terzo argomento

  per il momento ne consideriamo solo

  FD_CLOEXEC

FD_CLOEXEC	1	do close on exec
	0	do not close on exec

```
val = fcntl(fd, F_GETFD,0);
val |= FD_CLOEXEC;
fcntl(fd, F_SETFD, val);
```


fcntl(*filedes*, F_GETFL,0)

- restituisce i flag di stato per *filedes*

O_RDONLY,O_WRONLY,O_RDWR,O_APPEND, O_NONBLOCK, O_SYNC, …

- ATTENZIONE: per testare i flag per l'accesso dobbiamo usare la maschera *00000011* (O_ACCMODE in <fcntl.h>) con un AND bit a bit

…vediamo subito un esempio…

```
#include    <sys/types.h>     /*…dal programma in Fig. 3.10 */
#include    <fcntl.h>
#include    <unistd.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)
                printf("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");
              ……
```


…ATTENZIONE

- Quando si modificano i flag bisogna
  - Leggere il contenuto della parola che contiene i flag
    - Fare l'OR con il bit che si vuole attivare
    - Fare l'and con il negato del bit che si vuole disattivare
- Usare direttamente fcntl con F_SETFD o F_SETFL potrebbe disattivare alcuni flag settati in precedenza

```
#include    <sys/types.h>     /*…dal programma in Fig. 3.10 */
#include    <fcntl.h>
#include    <unistd.h>

void set_fl(int fd, int flags) /* flags:file status flags */
{
        int             val;

        if ((val = fcntl(fd, F_GETFL, 0)) < 0)
        {
                printf("\n ERROR: fcntl F_GETFL error\n");
                exit(0);
        }

        val |= flags;           /* turn on flags */

        if (fcntl(fd, F_SETFL, val) < 0)
        {
                printf("\nERROR: fcntl F_SETFL error\n");
                exit(0);
        }
}
```


Le funzioni sync

#include<unistd.h>

int sync(void);

tutti I buffer modificati vanno in coda per scrittura

int fsync(int fd);

attende che il file fd sia modificato compresi gli attributi

Utile per gestire DataBase

int fdatasync(int fd);

attende che il file fd sia modificato - solo dati

Return value: 0 se OK, -1 per errore.

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