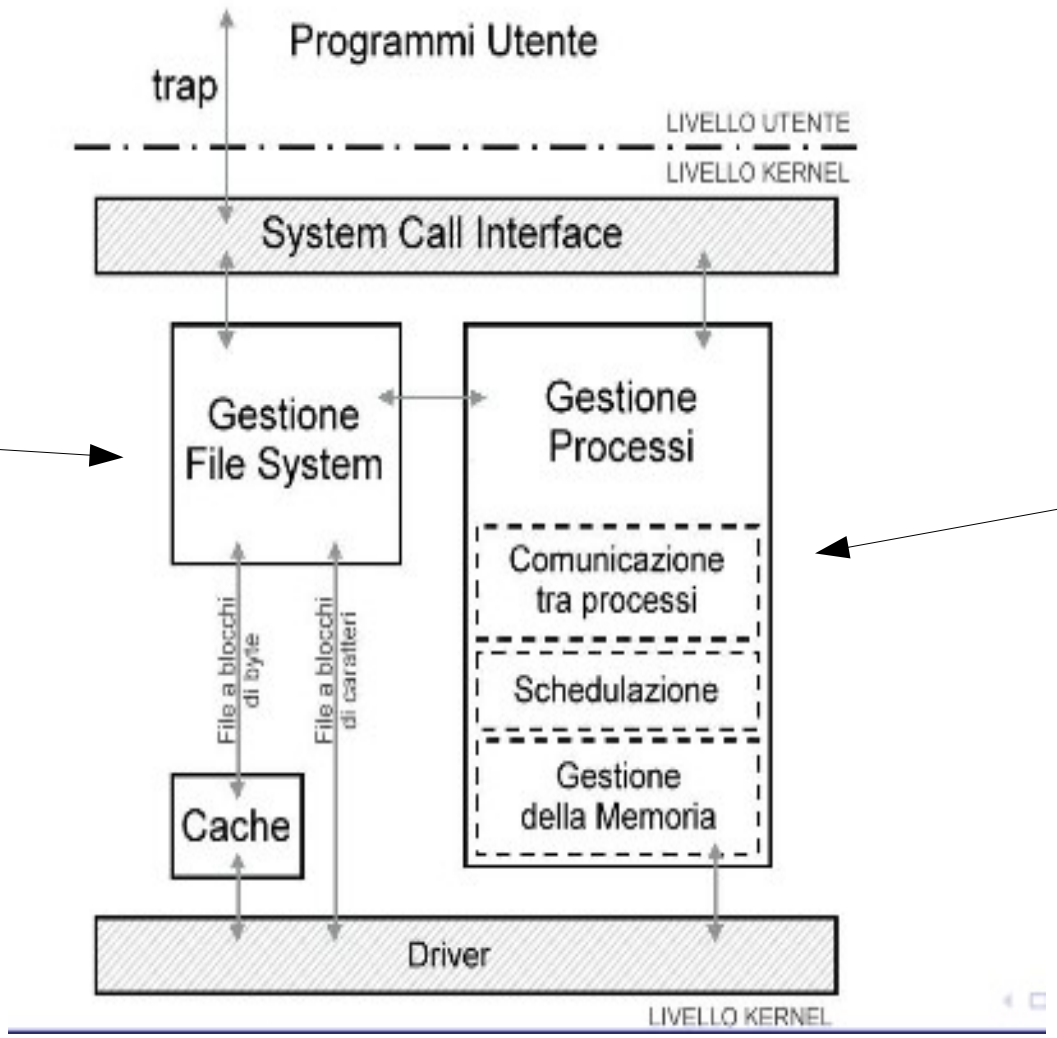
Programmazione di sistema in Linux: gestione dei file

E. Mumolo

Struttura di Linux

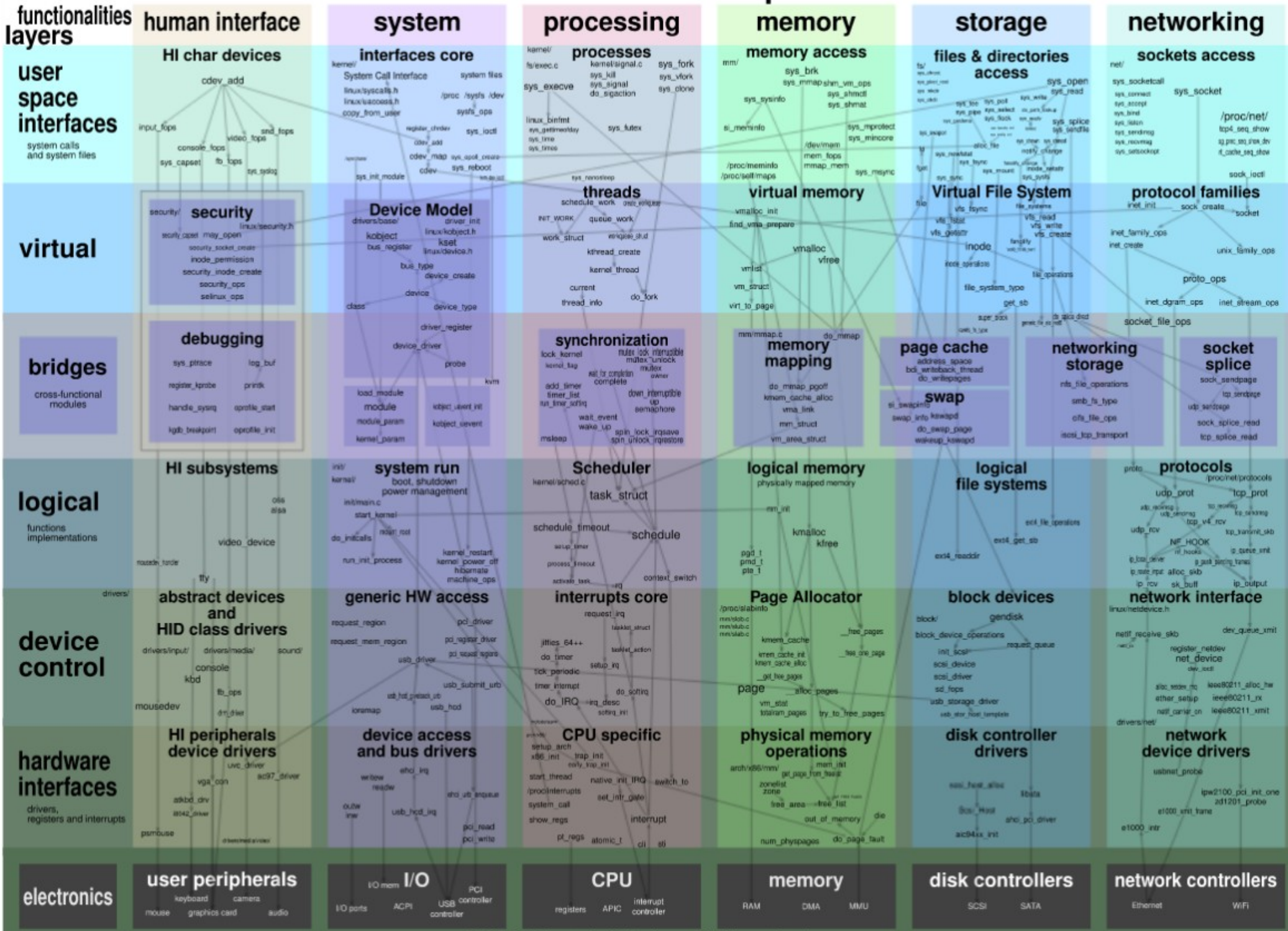


System Call per il file system

System Call per i processi

Hardware: dischi, CPU, monitor, tastiera, schede di rete, porte USB….

Linux kernel map



System Call (SC) per il file system

- SC che restituiscono descrittori di file:
  - open, dup, pipe, close
- SC che usano il Pathname:
  - open, stat, link, unlink, chdir, chroot, chown, chmod, mount, umount, mknode
- SC che assegnano Inode
  - mknode, link, ulink
- SC che effettuano I/O su file
  - read, write, lseek

Cambiamento filemode, proprietario

```
int chmod (const char* path, mode_t mode);
int fchmod (int filedes, mode_t mode);
```

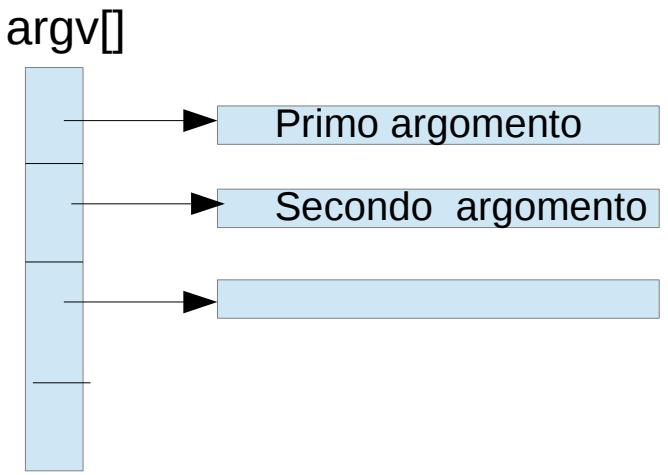
- Cambia i diritti di un file specificato dal pathname (chmod) o di un file aperto (fchmod)
- Per cambiare i diritti di un file, l'effective uid del processo deve essere uguale all'owner del file oppure deve essere uguale a root

```
int chown(char* pathname, uid_t owner, gid_t group);
int fchown(int filedes, uid_t owner, gid_t group);
int lchown(char* pathname,uid_t owner, gid_t group);
```

- Queste tre funzioni cambiano lo user id e il group id di un file
  - Nella prima, il file è specificato come pathname
  - Nella seconda, il file aperto è specificato da un file descriptor
  - Nella terza, si cambia il possessore del link simbolico, non del file stesso
- Restrizioni:
  - In alcuni sistemi, solo il superuser può cambiare l'owner di un file (per evitare problemi di quota)

Esempio di chmod

- Funzioni utilizzate:
  - `long int strtol(const char *str, char **endptr, int base)` : converte str in longint secondo la base
  - `int chmod(const char *pathname, mode_t mode);`




```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/stat.h>
int main(int argc, char **argv)
{
    char mode[] = "0777";
    char buf[100] = "/home/mumolo/test";
    int i;
    i = strtol(mode, 0, 8);
    if (chmod (argv[1],i) < 0)
    {
        fprintf(stderr, "%s: errore in chmod(%s, %s) - %d (%s)\n",
            argv[0], buf, mode, errno, strerror(errno));
        exit(1);
    }
    return(0);
}
```

Gestione file

- Un file per essere usato deve essere aperto (open)
- L'operazione open:
  - localizza il file nel file system attraverso il suo pathname
  - copia in memoria il descrittore del file (i-node)
  - **ritorna un intero non negativo (file descriptor)**
  - I file standard non devono essere aperti, perchè sono aperti dalla shell.
  - Sono associati ai file descriptor 0 (input), 1 (output) e 2 (error).
  - La close rende disponibile il file descriptor per ulteriori usi
- Un file può essere aperto più volte, e quindi avere più file descriptor associati contemporaneamente
- **Ogni apertura di un file appare come un nuovo elemento della file table**

Gestione file

- **Apertura del file:**

```
int open(const char *path,int oflag,mode_t mode);
```

- apre (o crea) il file specificato da **pathname** (assoluto o relativo), secondo la modalità specificata in **oflag**
- restituisce il file descriptor con il quale ci si riferirà al file successivamente (o -1 se errore)
- mode=file mode
- **Valori di oflag**
  - **O_RDONLY** read-only (0)
  - **O_WRONLY** write-only (1)
  - **O_RDWR** read and write (2)
  - Solo una di queste costanti può essere utilizzata in oflag
  - Altre costanti (che vanno aggiunte in or ad una di queste tre) permettono di definire alcuni comportamenti particolari

Gestione file

- Alcuni altri valori di oflag:
  - **O_APPEND** append
  - **O_CREAT** creazione del file
  - **O_SYNC** synchronous write

  → Se si specifica **O_CREAT**, è necessario specificare anche i permessi iniziali come terzo argomento

- **Esempi:**
  - `fd = open("testfile.txt", O_WRONLY | O_APPEND);`
  - `fd = open("testfile.txt", O_CREAT | O_WRONLY , 0666);`
  - `fd = open("testfile.txt", O_RDWR);`
  - `fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, mode);`

Gestione file

- **`int close(int filedes);`**
  - chiude il file descriptor **filedes**
  - restituisce l'esito dell'operazione (0 o -1)
  - Quando un processo termina, tutti i suoi file vengono comunque chiusi automaticamente

Strutture dati per i file

- **Ogni file aperto e' associato a:**
  - un "current file offset", la posizione attuale all'interno del file: è un valore non negativo che misura il numero di byte dall'inizio del file
  - operazioni **read/write** leggono dalla posizione attuale e incrementano il current file offset in avanti del numero di byte letti o scritti
- **Quando viene aperto un file**
  - Il current file offset viene posizionato a 0…
  - a meno che l'opzione **O_APPEND** non sia specificata

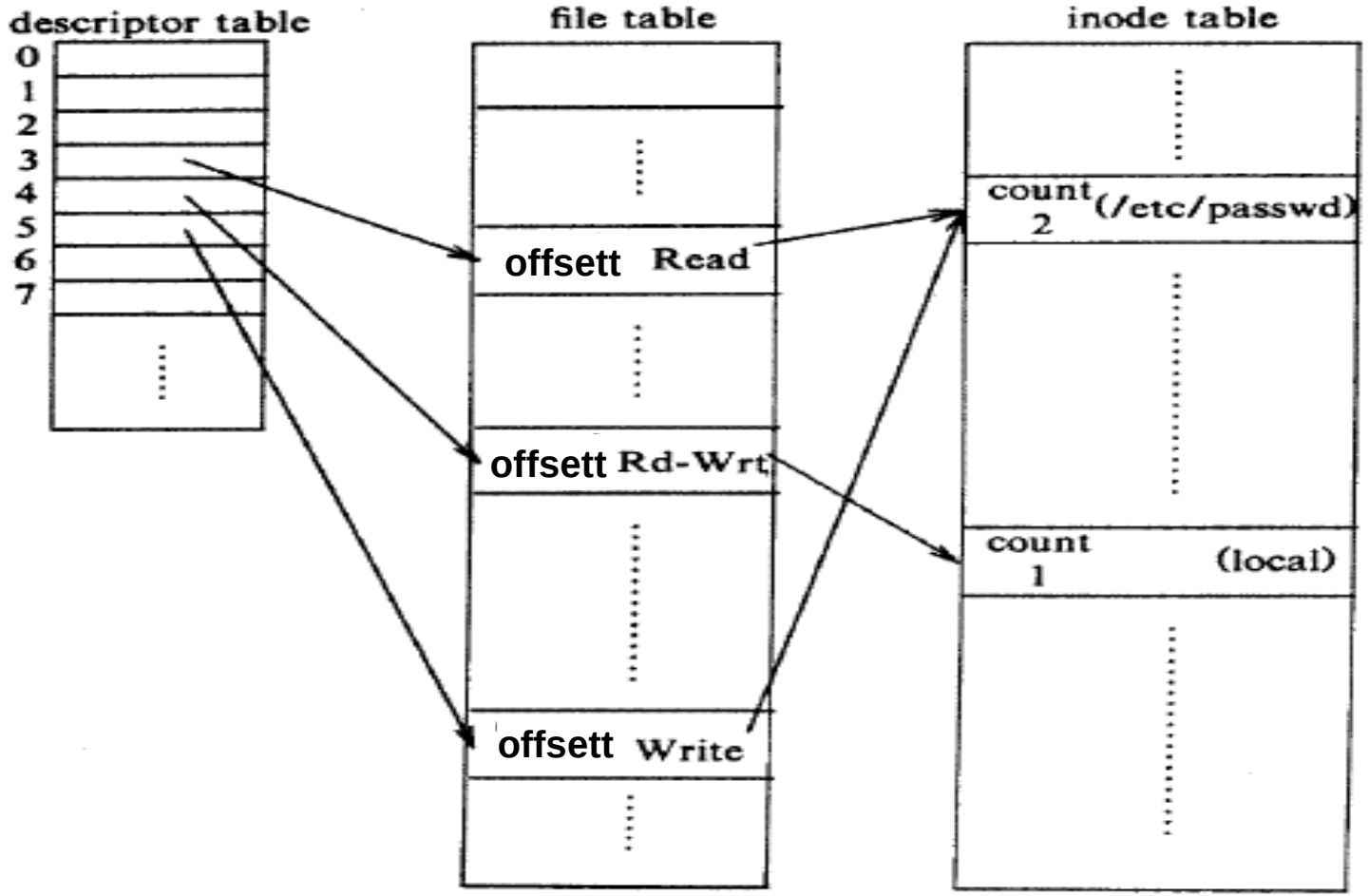
Strutture dati per i file

- User file descriptor table: personale di ogni processo
  - Contiene il descrittore ad un file aperto
  - I primi 3 sono automaticamente aperti:
    - 0 = standard input
    - 1 = output
    - 2 = standard error
- File table: comune a tutti I processi
- Inode table: tabella degli inode contenuta nei primi blocchi del disco

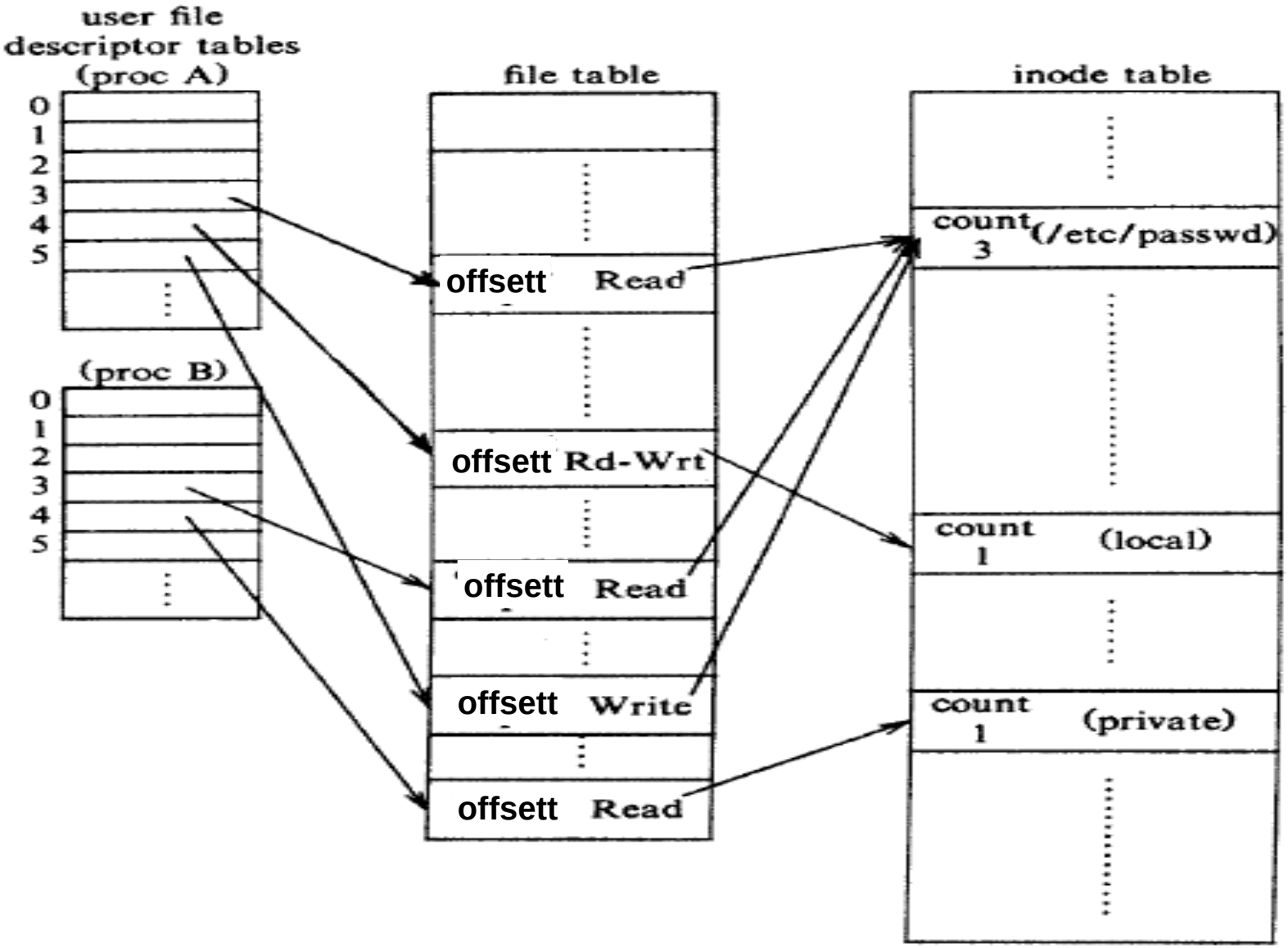
Strutture dati dopo tre open da parte di un processo

Codice:

```
Fd1 = open("/etc/passwd", O_RDONLY);
Fd2 = open ("local", 0 RDWR);
Fd3 = open(/etc/passwd", O WRONLY);
```

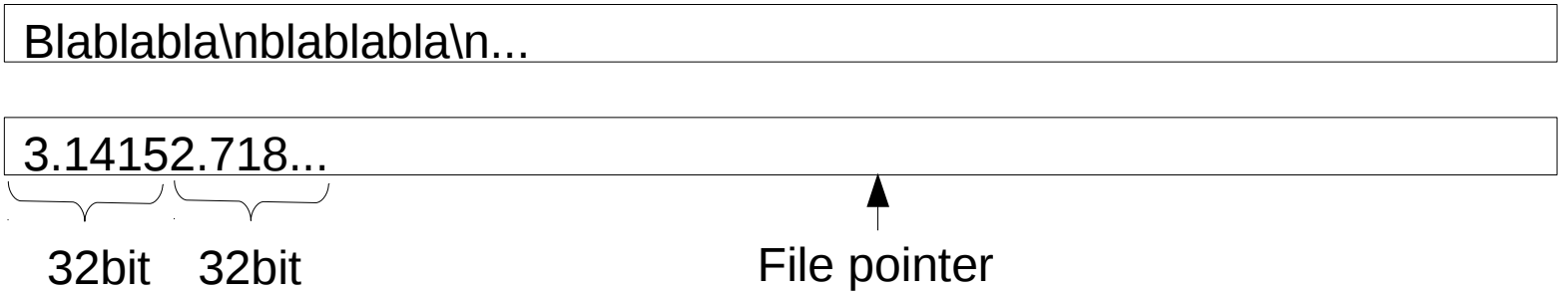


Strutture dati dopo tre open da parte di due processi



Struttura logica di un file

- Array di byte
  - Testo
  - Float
  - Strutture dati ...



- Le system call che agiscono sul file muovono un file pointer che punta al prossimo elemento

Gestione file

- **`off_t lseek(int filedes, off_t offset, int pos);`**
  - sposta la posizione corrente nel file **filedes** di **offset** bytes a partire dalla posizione specificata in **pos**:
    - **SEEK_SET** dall'inizio del file
    - **SEEK_CUR** dalla posizione corrente
    - **SEEK_END** dalla fine del file
    - restituisce la posizione corrente dopo la **lseek**, o -1 se errore

- **`lseek`** `non effettua alcuna operazione di I/O`

- In `/usr/include/stdio.h` c'è una linea:
  - `typedef long int off_t;`

- Dimensione file

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
int main (int argc, char **argv) {
   long p; int fd;
   if((fd = open(argv[1], O_RDWR, 0600)) < 0 ) {
         perror("errore in open\n"); return -1; }
   if ((p=lseek(fd, 0, SEEK_END)) == -1)
         printf ("errore in seek\n");
   else
         printf("dimensione del file in byte %ld\n", p);
   exit(0);
}
```

- Trova la posizione corrente in un file

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
int main (int argc, char **argv) {
     long p; int fd; char buf[10];

     if((fd = open(argv[1], O_RDWR, 0600)) < 0 ) {
               perror("errore in open\n"); return -1;
        }
     read(fd, buf, 6);
     if ((p=lseek(fd, 0, SEEK_CUR)) == -1)
             printf ("errore in seek\n");
     else
             printf("posizione corrente in byte %ld\n", p);
     exit(0);
}
```

Gestione file

`ssize_t read(int filedes,void *buf, size_t nbyte);`

- legge in ***buf** una sequenza di **nbyte** byte dalla posizione corrente del file **filedes**
- aggiorna la posizione corrente
- restituisce il numero di bytes effettivamente letti, o -1 se errore
- Esistono un certo numero di casi in cui il numero di byte letti e' inferiore al numero di byte richiesti:
  - Fine di un file regolare
  - Per letture da stream provenienti dalla rete
  - Per letture da terminale
  - etc.

`ssize_t write(int filedes,const void *buf,size_t nbyte);`

- scrive da ***buf** una sequenza di **nbyte** byte dalla posizione corrente del file **filedes**
- aggiorna la posizione corrente
- restituisce il numero di bytes effettivamente scritti, o -1 se errore

Pseudocodice Algoritmo di read()

`read(int filedes,void *buf, size_t nbyte);`

```
int read(filedes, *buf, nbyte){
   Legge il numero di  inode dalla FileTable;
   Verifica se il file è accessibile;
   nbyte=numero di byte da leggere;
   Legge il file offsett dalla filetable;
   while (nbyte non soddisfatto){
      Converte l'offset al file in numero del blocco dati;
      Calcola l'offsett nel blocco;
      Legge il blocco dati;
      Copia I dati dallo spazio kernel alo spazio utente;
      Aggiorna offsett, numero byte da leggere;
   }
   Aggiorna il file offsett nella filetable;
   return (numero totale di byte letti);
}
```

Legge i dati blocco per blocco

Attenzione: file a blocchi!

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/stat.h>
void main()
{
      int i; char c;
      for(i=0;i<10;i++){
            read(0, &c, 1);
            write(1, &c, 1);
}
```

Legge caratteri dall'unita' 0 e li scrive sulla 1

==========0

```
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>

void main()
{
      int i; char c; int fd1, fd2;
      fd1=open("/dev/tty",O_RDONLY); fd2=open("/dev/tty",O_WRONLY);
      for(i=0;i<10;i++){
            read(fd1, &c, 1);
            write(fd2, &c, 1);
}
```

Stessa cosa ma usando il file /dev/tty

- Scrive argv[3] in argv[1] in posizione argv[2]

```
#define _FILE_OFFSET_BITS 64
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>

int main (int argc, char **argv) {

        int fd = 0;
        off_t off = 0;

        if((fd = open(argv[1], O_RDWR, 0600)) < 0 ) {
                perror("errore in open\n"); return -1;
        }

        off = atol(argv[2]);

        if(lseek(fd, off, SEEK_SET) < 0) {
                perror("errore in lseek\n"); return -1;
        }

        if (write(fd, argv[3], strlen(argv[3])) < 0 ) {
                perror("errore in write\n");   return (-1);
        }

        close(fd);

        return 0;
}
```

Esempio: mycat.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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) {
            perror("write error");
            exit(1);
        }
    if (n < 0) { perror("read error"); exit(1); }
    exit(0);
}
```

Copia file vs. buffersize

BUFFSIZE	User CPU (seconds)	System CPU (seconds)	Clock time (seconds)	# loops
1	23.8	397.9	423.4	1468802
2	12.3	202.0	215.2	734401
4	6.1	100.6	107.2	367201
8	3.0	50.7	54.0	183601
16	1.5	25.3	27.0	91801
32	0.7	12.8	13.7	45901
64	0.3	6.6	7.0	22950
128	0.2	3.3	3.6	11475
256	0.1	1.8	1.9	5738
512	0.0	1.0	1.1	2869
1024	0.0	0.6	0.6	1435
2048	0.0	0.4	0.4	718
4096	0.0	0.4	0.4	359
8192	0.0	0.3	0.3	180
16384	0.0	0.3	0.3	90
32768	0.0	0.3	0.3	45
65536	0.0	0.3	0.3	23
131072	0.0	0.3	0.3	12

- Blocco disco di 512 byte
- Perchè questo comportamento? La risposta sta nella lettura dei dati della read()

Lettura/scrittura formattata da/su file ASCII
NON solo chiamate di sistema ma funzioni

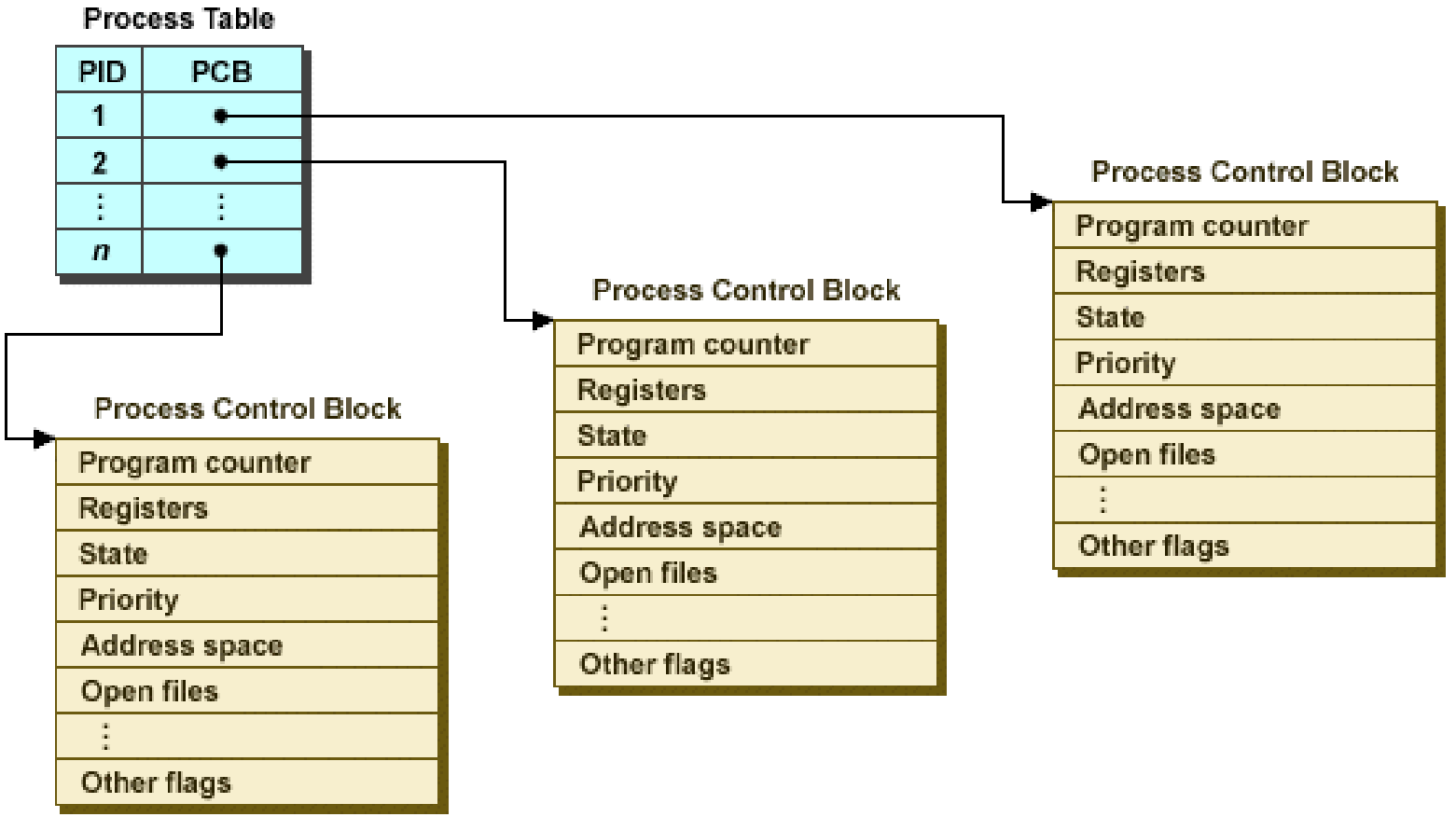
- Si introduce un file pointer di tipo FILE
```
FILE *fp;
```
- Apertura del file
```
FILE *fopen(const char *path, const char *mode);
```
- Lettura formattata da file
```
int fscanf(FILE *stream, const char *format, …)
```
- Lettura di un file ascii linea per linea
```
char *fgets(char *str, int n, FILE *stream)
```
- Scrittura formattata su file
```
int fprintf(FILE *stream, const char *format, …)
```
- Chiusura file
```
int fclose(FILE *stream);
```

Esempio

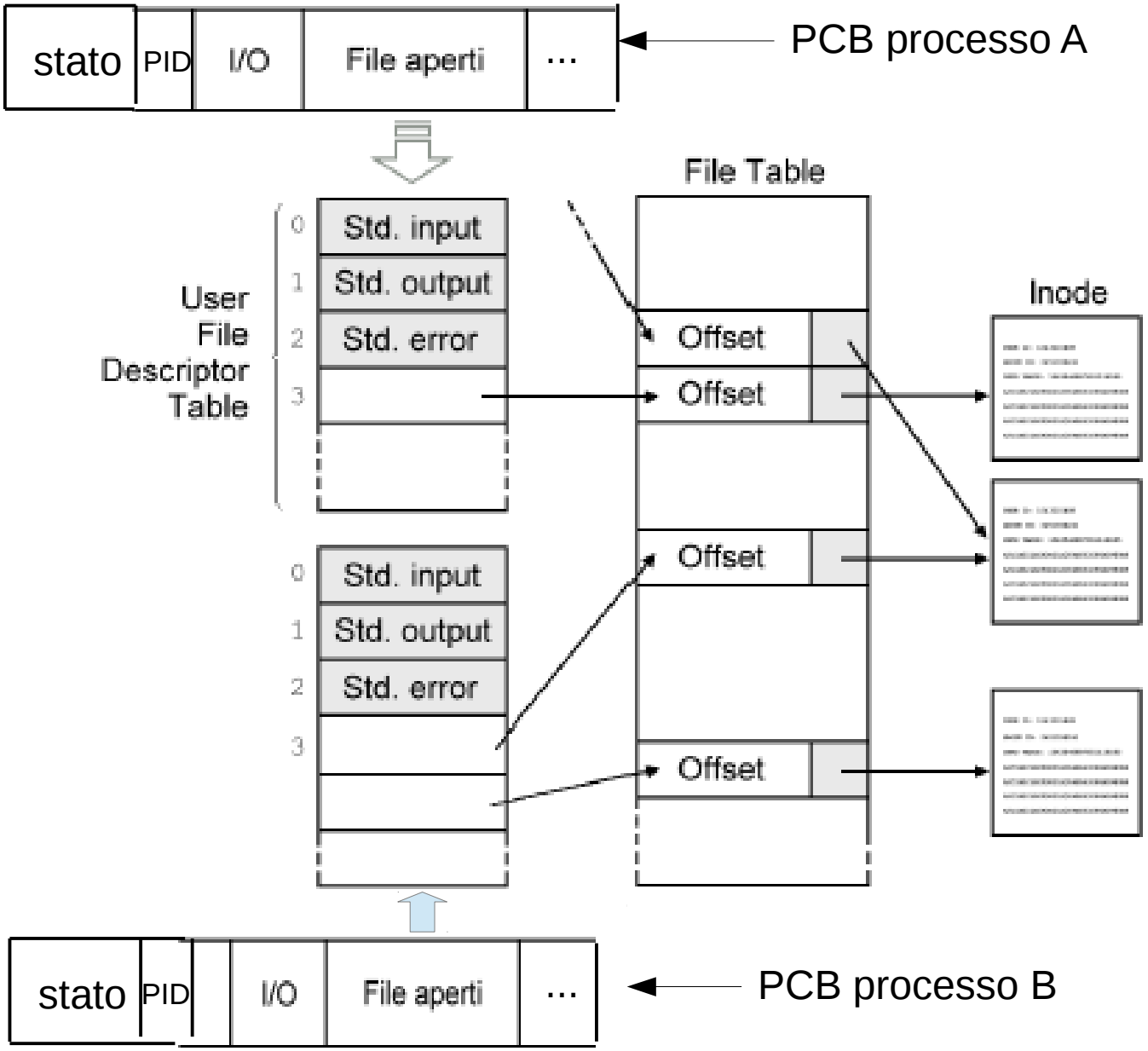
Lettura di 10 righe dal file testmed.txt e memorizzazione in un array 2D di stringhe

```
char **data=malloc(1000*sizeof(char*));
fp=fopen("testmed.txt","r"));
for(i=0;i<10;i++){
        fgets(testo, size, fp);
        data[i]=malloc(strlen(testo));
        strcpy(data[i],testo);
}
close (fd);
```

Visione d'insieme



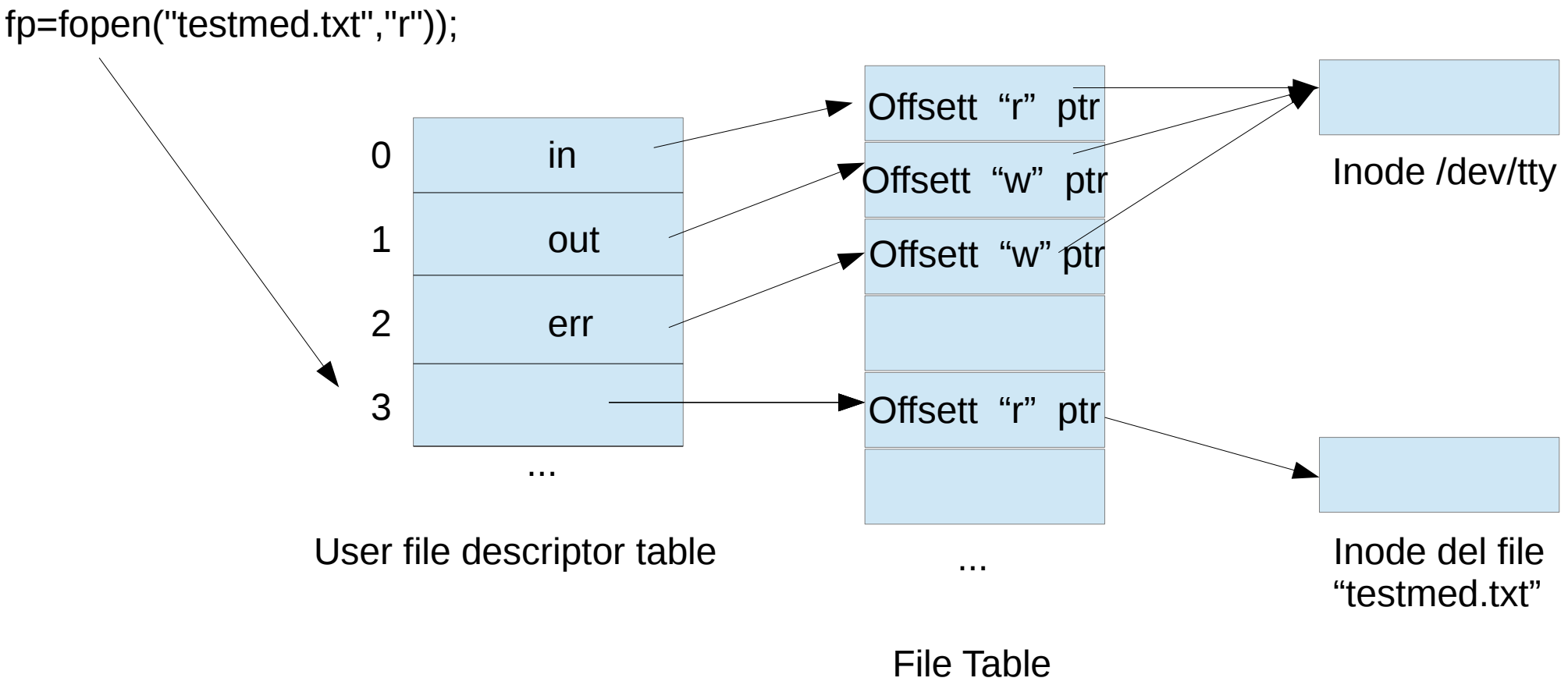
Visione d'insieme



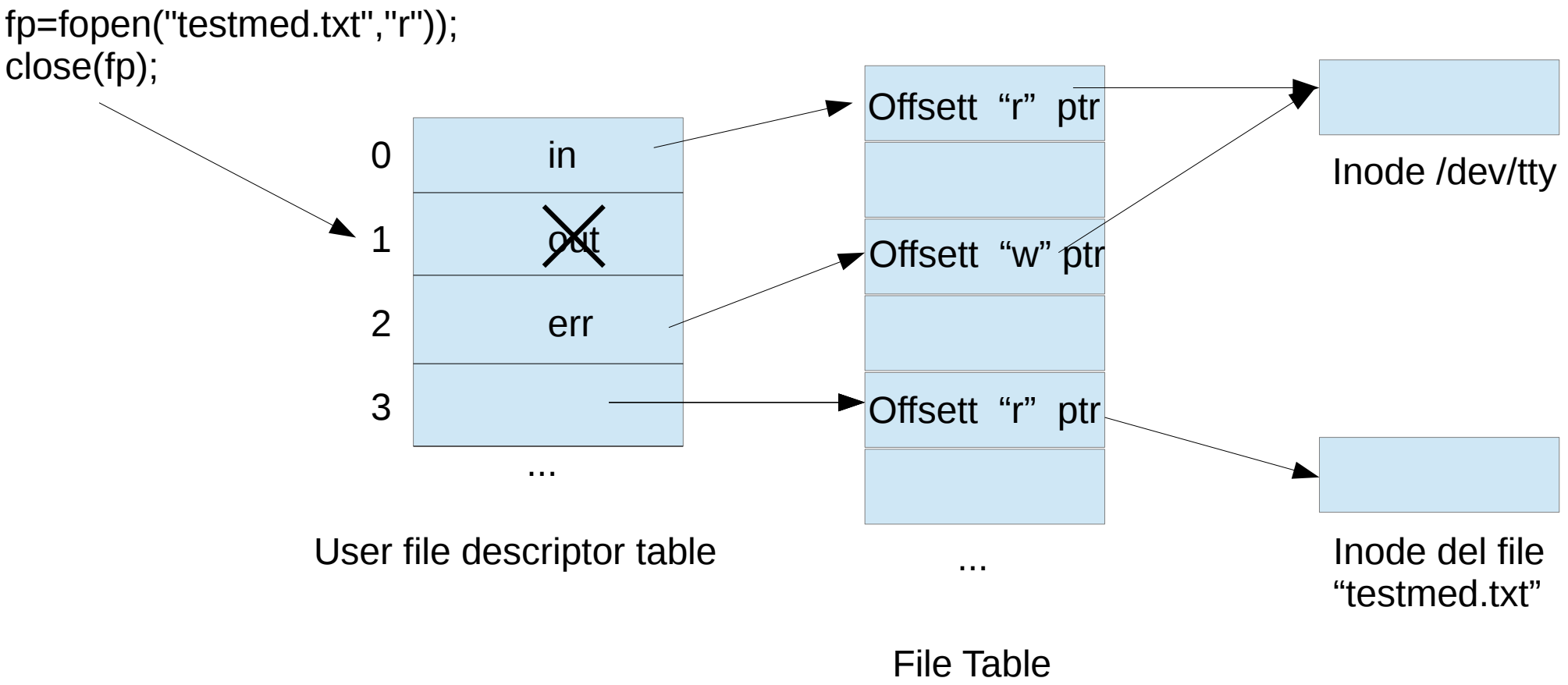
Duplicazione descrittori

- **Funzione dup: int dup(int oldfd);**
  - Seleziona il più basso file descriptor libero della tabella dei file descriptor
  - Assegna la nuova file descriptor entry al file descriptor selezionato
  - Ritorna il file descriptor selezionato
- **Funzione dup2: int dup2(int oldfd, int newfd);**
- Con **dup2**, specifichiamo il valore del nuovo descrittore come argomento **filedes2**
- Se **filedes2** è già aperto, viene chiuso e sostituito con il descrittore duplicato
- Ritorna il file descriptor selezionato

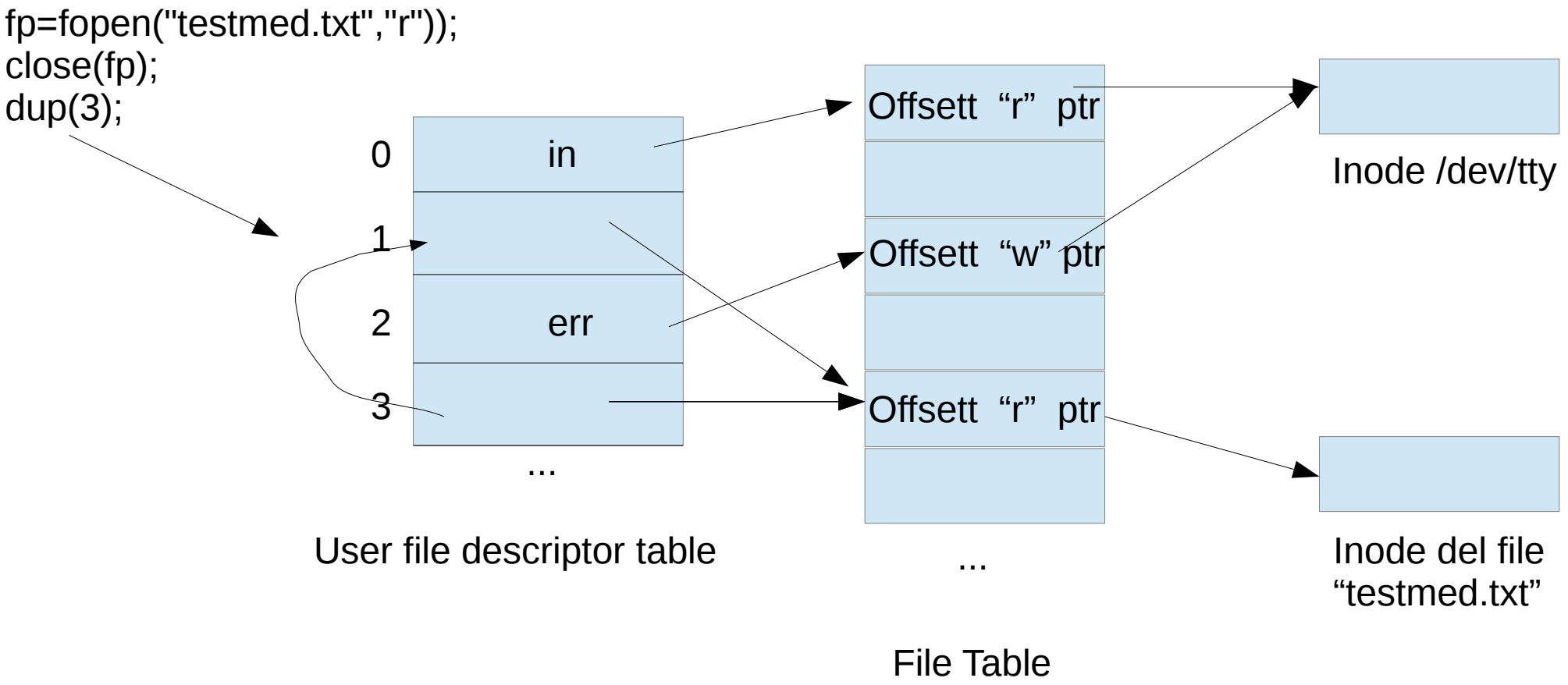
Duplicazione descrittori



Duplicazione descrittori

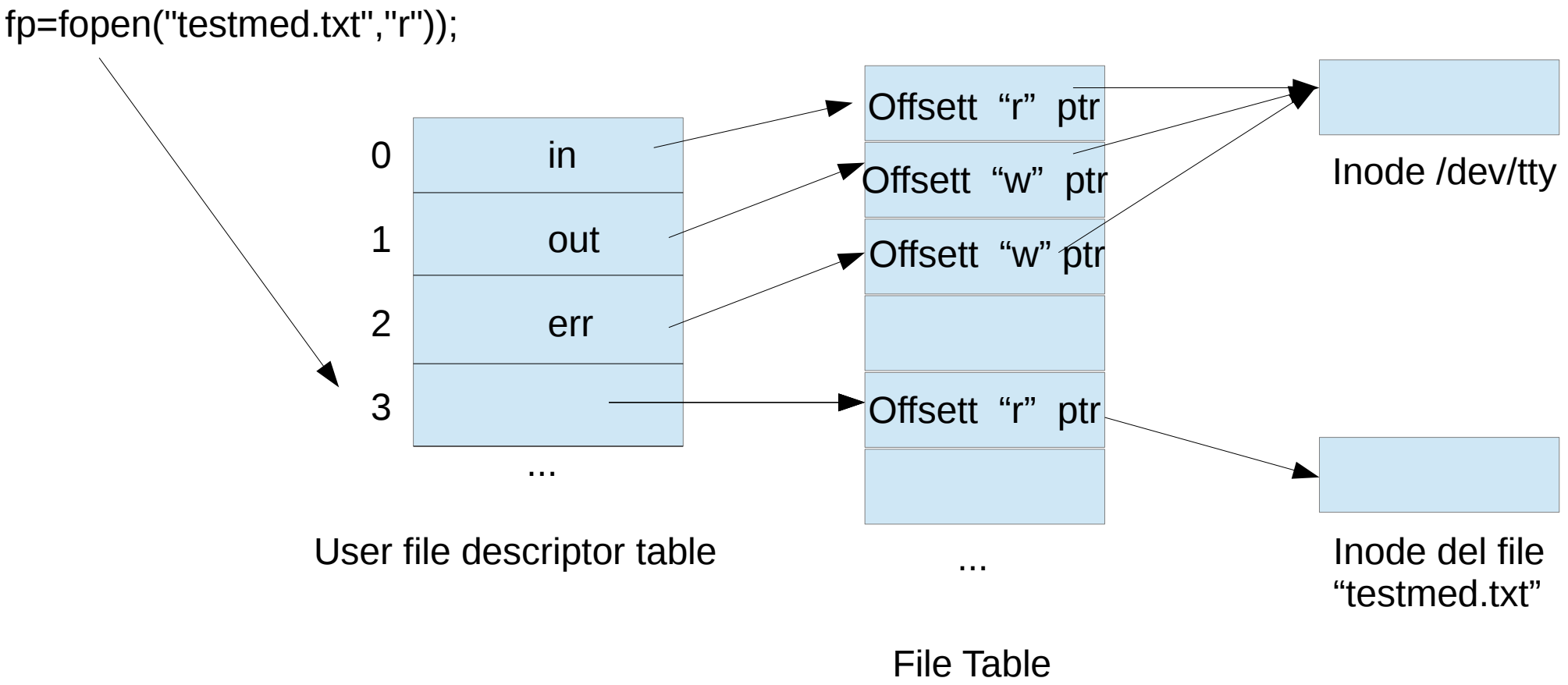


Duplicazione descrittori



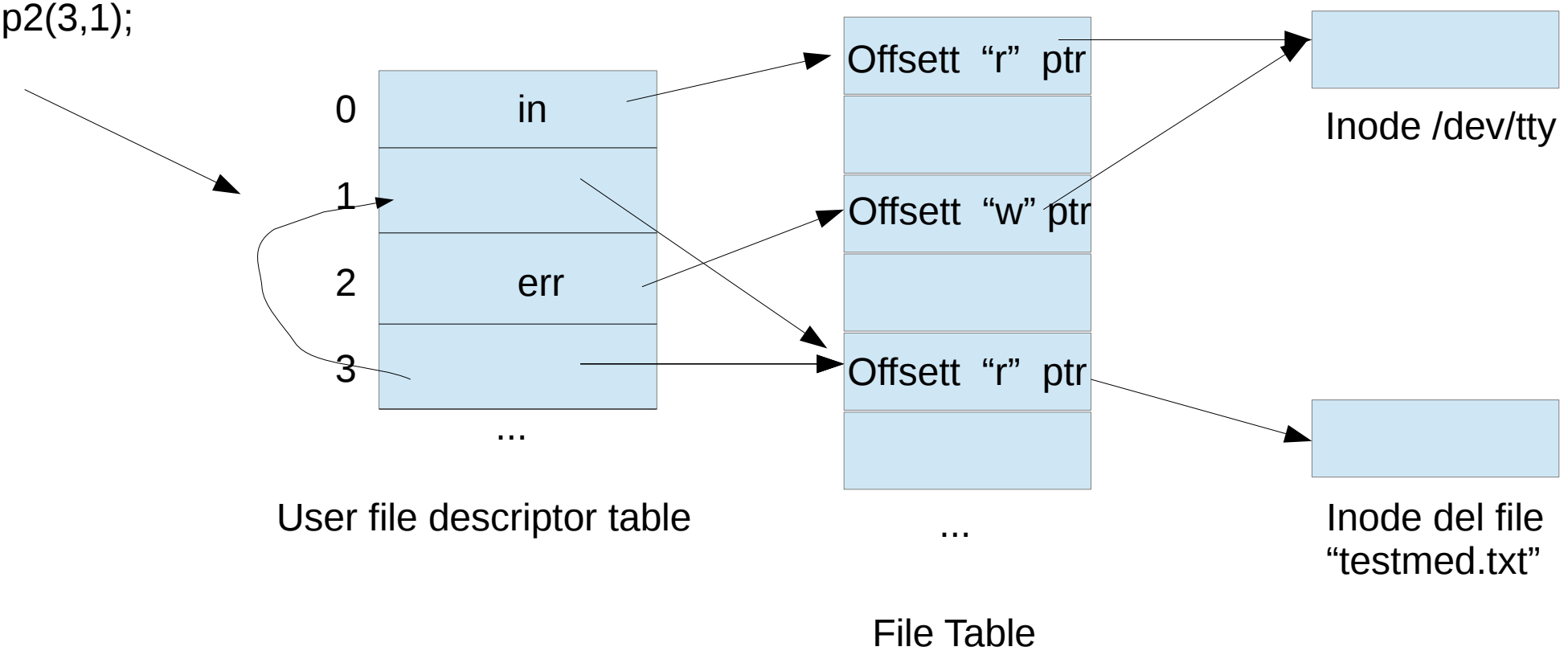
Oppure...

Duplicazione descrittori



Duplicazione descrittori

fp=fopen("testmed.txt","r"));
dup2(3,1);



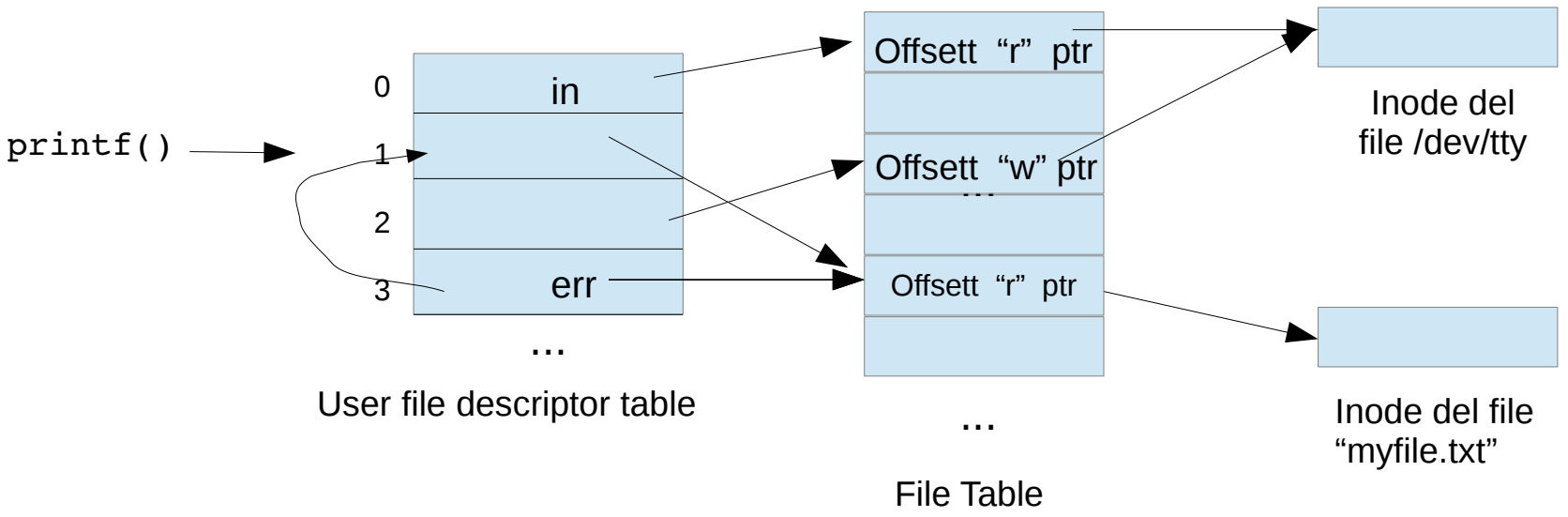
Duplicazione descrittori

==========DUP==========

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
    int file = open("myfile.txt",  O_CREAT|O_WRONLY, 0766);
    if(file < 0)  perror("open");

    close(1);
    if(dup(file) < 0)  perror("dup");;
    printf("dove verra' scritto questo primo testo");
    printf("secondo  testo");
    return 0;
}
```

Redirezione!!!



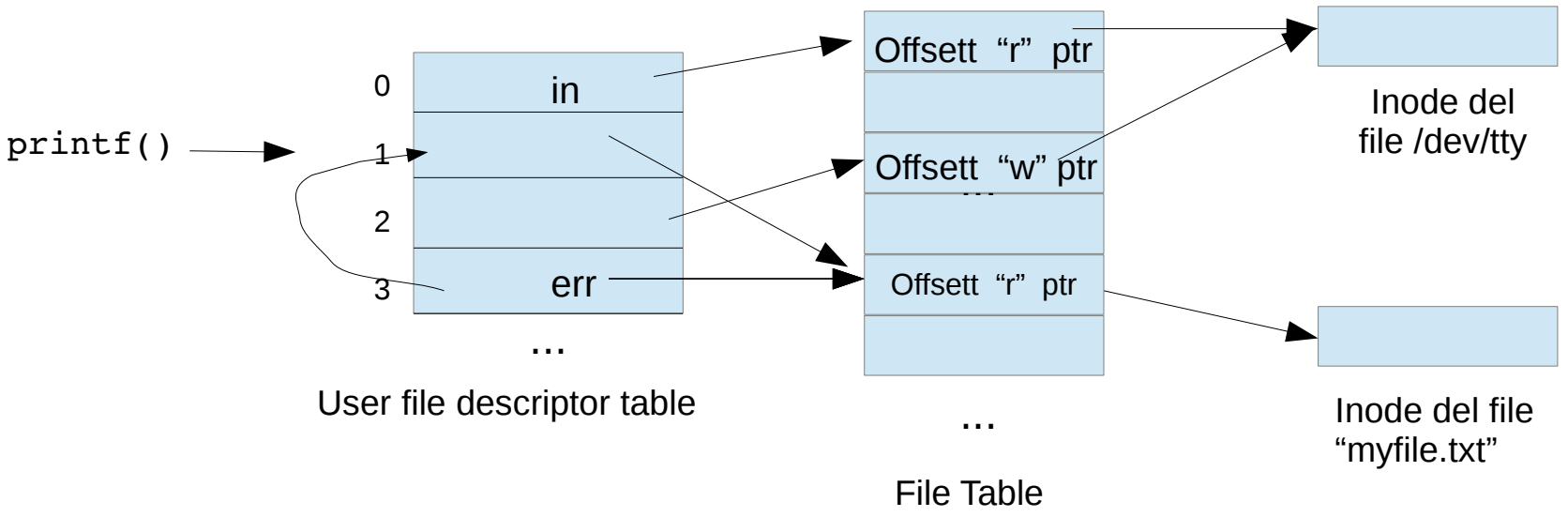
Duplicazione descrittori

==========DUP2==========

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
    int file = open("myfile.txt",  O_CREAT|O_WRONLY, 0766);
    if(file < 0)  perror("open");

    if(dup2(file,1) < 0)  perror("dup2");;
    printf("dove verra' scritto questo primo testo?\n");
    printf("secondo  testo");
    return 0;
}
```

Redirezione!!!



Directory

`int mkdir(char* path, mode_t);`

- Crea una nuova directory vuota dal path specificato
- Modalità di accesso:
  - I permessi specificati da mode_t vengono modificati dalla maschera specificata da umask
  - E' necessario specificare i diritti di esecuzione (search)

`int rmdir(char* path);`

- Rimuove la directory vuota specificata da path
- Se il link count della directory scende a zero, la directory viene effettivamente rimossa; altrimenti si rimuove la directory

`int chdir(char* path);`

`int fchdir(int filedes);`

  - Cambia la directory corrente associata al processo, utilizzando un pathname oppure un file descriptor

File status

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

int stat(const char *path, struct stat *buf);  // path
int fstat(int fd, struct stat *buf);           // fp
int lstat(const char *path, struct stat *buf); //se path è un link soft, stato del link
```

- Tutte queste Sys Call tornano un ptr alla struttura stat; tornano -1 se il file non esiste

```
stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

- Crea una directory se non esiste

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

struct stat st = {0};
int main()
{
    if (stat("/home/mumolo/mysub/", &st) == -1) {
        mkdir("/home/mumolo/mysub/", 0700);
    }
}
```

System()

- Primo modo (semplificato) per creare un processo
- Fornisce un'interfaccia di shell all'interno di un processo

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
const char * const path = "/home/mumolo/sub";
const char * const file = "FileDiTest.txt";
int main () {
   printf ("sto cambiando directory  <%s>\n", path);
   if (chdir (path) == -1) { printf ("errore in chdir - %d\n", strerror (errno));}
   else {
       printf ("chdir ok\n");
       printf ("contenuto di '%s'\n\n", path);
       system ("ls -l");   printf ("\n");
       printf ("contenuto del file '%s'\n", file);
       system ("cat FileDiTest.txt");
   }
   return 0;
}
```

Hard link

int link(char* oldpath, char* newpath);

- crea un nuovo nome ad un file esistente
- operazione atomica: aggiunge la directory entry e aumenta il numero di link per l'inode identificato da **oldpath**

- errori:
  - **oldpath** non esiste
  - **newpath** esiste già
  - solo root può creare un hard link ad una directory (per evitare di creare loop, che possono causare problemi)
  - **oldpath** e **newpath** appartengono a file system diversi

- utilizzato dal comando **ln**

Esempio di hard link

```
#include <stdio.h>
 int main()
 {
    if ((link("filevecchio.txt", "filenuovo.txt")) == -1)
       {
            perror(" ");
            exit (1);         /* se errore ritorna un numero diverso da zero */
       }
    exit(0);
 }
```

Link simbolici

```
int symlink(char* oldpath, char* newpath);
```

- crea un nuovo file **newpath** con un link simbolico che punta al file specificato da **oldpath**
- nota:
  - **oldpath** e **newpath** non devono risiedere necessariamente nello stesso file system
- Comando di shell `ln -l`

Cancellazione file

int unlink(char* path);

- decrementa di uno il numero di link del file
- un file può essere effettivamente cancellato dall'hard disk quando il suo conteggio raggiunge 0
- utilizzato dal comando **rm**

Qualche Esercizio

Si descrivano le tabelle utilizzate dal file system di Linux nel caso che un programma apra il file ‘pippo.txt’ tre volte in modalita’ diverse (rispettivamente scrittura, scrittura e lettura) dopo che ogni apertura del file seguano rispettivamente una scrittura di 10 byte, una scrittura di 5 byte ed una lettura di 10 byte. Se il file contiene la stringa “abcdefghilmno”, come si modifica il file? Quali sono I 10 byte letti?

Scrive sul monitor I primi 100 char del file testmed.txt con read/write

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
main()
{
    char testo[100], new[100];
    int i,fd,size=100;


    if  ((fd=open("testmed.txt",O_RDONLY))<0) {printf ("errore apertura file\n"); exit (1);}

    read(fd,testo,sizeof(testo));

    write(1,testo,sizeof(testo));

    close (fd);
}
```

Scrive sul monitor la prima riga del file testmed.txt con fgets/write

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
main()
{
    char testo[100], new[100];
    int i,fd,size=100;
    FILE *fp;


    if  (fp=fopen("testmed.txt","r"))<0) {printf ("errore apertura file\n"); exit (1);}

    fgets(testo, sizeof(testo), fp);

    write(1,testo,sizeof(testo));

    close (fd);
}
```

Scrive i primi 100 char del file dato in linea con read/write

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
void main(int argc, char *argv[])
{
    char testo[100]={0}, new[100]={0};
    int i,fd,size=100;
    char p, *p1=testo, *p2=new;
    char nome[20];

    strcpy(nome, argv[1]);
    p='\n';                                 /* assegno a p il carattere <return> o End of Line */

    if  ((fd=open(nome,O_RDONLY))<0) {printf ("errore apertura file\n"); exit (1);}

    read(fd,testo,sizeof(testo));      /* leggo 100 byte */

    for(i=0; *p1!='\n';i++) *p2++=*p1++;  /*copio in new[] i char di testo[] prima del return */
    *p2='\0';   /* terminatore di stringa */

    write(1,new,sizeof(new));          /* scrive sulla consolle la stringa ascii */
    write(1,&p,1);                     /* scrive End of Line sulla consolle */

    close (fd);
}
```

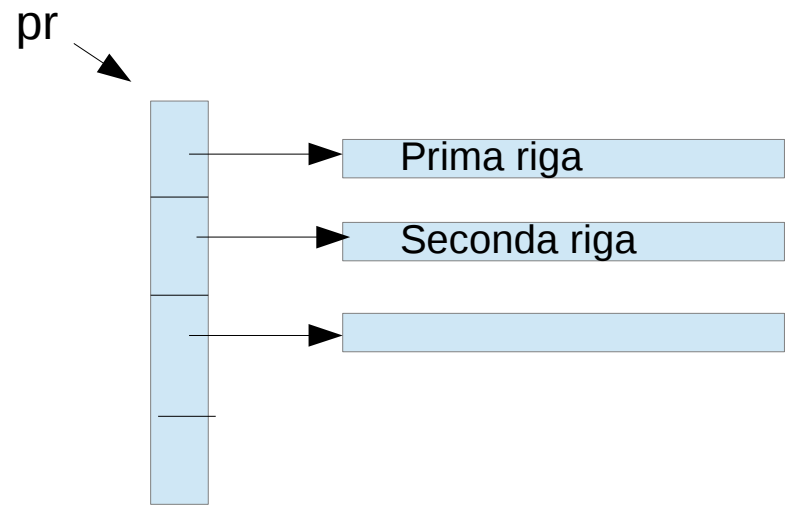
```
$gcc prog.c o prog
$./prog testmed.txt
```

Copia le prime N righe del file testmed.txt in un array di stringhe

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
void main(int argc, char *argv[])
{
    char nome[20]; FILE *fp; int i; char linea[100];
    typedef struct{
        int nr;
        char **pr;
    } tipo;
    tipo testo;

     strcpy(nome,argv[1]);
     testo.nr=atoi(argv[2]); //il nr di righe

    fp=fopen(nome, "r");
    testo.pr=(char **)malloc(testo.nr*sizeof(char *));
    for(i=0;i<testo.nr;i++){
        fgets(linea, 100, fp);
        testo.pr[i]=(char *)malloc(strlen(linea));
        strcpy(testo.pr[i],linea);
    }
    for(i=0;i<testo.nr;i++)
        printf("%s",testo.pr[i]);
}
```



Comando mytee -a file

- Il comando tee file copia lo stdin in stdout . In più scrive lo stdout sul file
- Esempio: `$ls -1 *.txt | wc -l | tee conta.txt`
- `L'opzione -a accoda`

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

int main(int argc, char **argv) {
   FILE *fp;

   int c = fgetc(stdin);

   fp = fopen(argv[2], "a+");
     if(fp==NULL)      {
           printf("Errore! Il file di origine non esistente!\n");
           return 1;
      }

   if ( argc!=3 || strcmp(argv[1], "-a") != 0){
       printf("Errore! Sintassi errata. Sintassi: ./tee -a [file] \n");
       exit(1);
   }
   while (c != EOF) {
      putchar(c);
      fputc(c, fp);
      c = getchar();
   }
   fclose (fp);
   return 0;
}
```