```
#include <stdio.h>
/* pipetest.c */
main()
{
        int pfd[2], nread;
        char s[100];

        if(pipe(pfd) == -1)
          syserr("pipe");
        if (write(pfd[1], "hello", 5) == -1)
          syserr("write");
        switch(nread = read(pfd[0], s, sizeof(s))){
        case -1 : syserr("read");
        case 0 : fatal("EOF");
        default: printf("read %d bytes: %s\n", nread, s);
        }
}


#include <stdio.h>
/* pipewrite.c */
main()
{
        int pfd[2];
        char fdstr[10];
        if (pipe(pfd) == -1)
                syserr("pipe");
        switch (fork()) {
        case -1: syserr("fork");
        case 0:
          if(close(pfd[1]) == -1)  /* il figlio chiude il writing end */
            syserr("close");
          sprintf(fdstr,"%d",pfd[0]);
          execlp("./pread","pread",fdstr,NULL);
          syserr("execlp");
        }
        if (close(pfd[0]) == -1)    /* il padre chiude il reading end */
           syserr("close2");
        sleep(2);

        if (write(pfd[1], "hello", 5) == -1)
           syserr("write");
}


#include <stdio.h>
/* piperead.c */
main(int argc, char **argv)
{
        int fd, nread;
        char s[100];

        fd = atoi(argv[1]);
        printf("sto leggendo il descrittore del file %d\n", fd);
        switch (nread = read(fd, s, sizeof(s))) {
        case -1: syserr("read");
        case 0: fatal("EOF");
        default: printf("ho letto %d bytes dalla pipe: %s\n", nread,s);
        }
}
```

```
#include <stdio.h>
/* pipewrite2.c */
main()
{
     int pfd[2];

     if (pipe(pfd) == -1)
               syserr("pipe");
     switch (fork()) {
       case -1: syserr("fork");
       case 0:
         if(close(0) == -1)  syserr("close");
         if (dup(pfd[0]) != 0) fatal("dup");
         if(close(pfd[0]) == -1 || close(pfd[1]) == -1)
            syserr("close2");
         execlp("cat","cat", NULL);
         syserr("execl");
     }
     if (close(pfd[0]) == -1) syserr("close3");
     sleep(2);
     if (write(pfd[1],"Questa stringa e' stampata da cat\n",37) == -1)
          syserr("write");
     if (close(pfd[1]) == -1) syserr("close4");
}

#include <stdio.h>
/* pipewho_wc.c */
main()
{
        int pfd[2];

        if(pipe(pfd) == -1) syserr("pipe");
        switch(fork()) {
        case -1: syserr("fork");
        case 0:
                if (close(1) == -1)   syserr("close");
                if (dup(pfd[1]) != 1) fatal("dup");
                if (close(pfd[0]) == -1 || close(pfd[1]) == -1)
                  syserr("close2");
                execlp("who", "who", NULL);
                syserr("execl");
        }
        switch(fork()) {
        case -1:
          syserr("fork");
        case 0:
          if(close(0) == -1) syserr("close3");
          if (dup(pfd[0]) != 0) fatal("dup2");
          if (close(pfd[0]) == -1 || close(pfd[1]) == -1)
            syserr("close4");
          execlp("wc", "wc", NULL);
          syserr("execl2");
        }
        if (close(pfd[0]) == -1 || close(pfd[1]) == -1)
          syserr("close5");
        while(wait(NULL) != -1) ;
}
```

```
#include <stdio.h>
/* pipe3.c */
main()
{
        int pfd[2], pfd1[2];
        char cmd[10], cmd1[10], cmd2[10];

        printf("mette in pipe tre comandi UNIX.\n");
        printf("Dare i tre comandi :");
        scanf("%s %s %s",cmd,cmd1,cmd2);

        if(pipe(pfd) == -1)     syserr("pipe");
        if(pipe(pfd1) == -1)    syserr("pipe");

        switch(fork()) {
        case -1: syserr("fork");
        case 0:
                if (close(0) == -1) syserr("close_wc");
                if (dup(pfd1[0]) != 0) fatal("dup_wc");
                if ( close(pfd1[0]) == -1 || close(pfd1[1]) == -1 ||
                      close(pfd[0]) == -1 || close(pfd[1]) == -1 )
                           syserr("close2_wc");

                execlp(cmd2, cmd2, NULL);
                syserr("execwc");
        }

        switch(fork()) {
        case -1:
          syserr("fork");
        case 0:
          if (close(0) == -1) syserr("close0_sort");
          if (dup(pfd[0]) != 0) fatal("dup0_sort");
          if (close(1) == -1) syserr("close1_sort");
          if (dup(pfd1[1]) != 1) fatal("dup1_sort");
          if ( close(pfd1[0]) == -1 || close(pfd1[1]) == -1 ||
               close(pfd[0]) == -1 || close(pfd[1]) == -1 )
                        syserr("close2_sort");
          printf("attivo sort\n");
          execlp(cmd1,cmd1, NULL);
          syserr("execlsort");
        }

        switch(fork()) {
        case -1:
           syserr("fork44");
        case 0:
           if (close(1) == -1)   syserr("close_ps");
           if (dup(pfd[1]) != 1) fatal("dup_ps");
           if (close(pfd1[0]) == -1 || close(pfd1[1]) == -1)
              syserr("close1_ps");
           execlp(cmd,cmd, NULL);
           syserr("execlps");
        }
         if (close(pfd1[0]) == -1 || close(pfd1[1]) == -1
           || close(pfd[0]) == -1 || close(pfd[1]) == -1)
                     syserr("close5");
        while(wait(NULL) != -1);
}
```

```
/* prova con pipe bidirez. */
/* pipebidir.c */
main()
{
        int pfdout[2], pfdin[2], fd, nread;
        char buf[512];

        pipe(pfdin); pipe(pfdout);

        switch (fork()){
        case -1: syserr("fork");
        case 0: close(0); dup(pfdout[0]);
                close(1); dup(pfdin[1]);
                close(pfdout[0]); close(pfdin[1]);
                close(pfdout[1]); close(pfdin[0]);
                execlp("sort","sort",NULL);
                syserr("execl");
        }
        close(pfdout[0]); close(pfdin[1]);
        if((fd=open("dati",0)) == -1) syserr("open");
        while((nread=read(fd,buf,sizeof(buf))) != 0)
                if(write(pfdout[1],buf,nread) == -1) syserr("write");
        close(fd); close(pfdout[1]);
        while((nread=read(pfdin[0],buf,sizeof(buf))) != 0)
                write(1,buf,nread);
        close(pfdin[0]);
}
```

```
/************************Coprocessori */
#include <stdio.h>
/* add.c */
int main()
{
        int n, int1, int2;
        char linea[100];

        read(0, linea, 100);
        sscanf(linea, "%d %d", &int1, &int2);
        sprintf(linea,"%d %d \n", int1+int2, int1*int2);
        write(1,linea,strlen(linea));
        exit(0);
}
/************************************/
#include        <stdio.h>
#include        <unistd.h>
#define MAXLINE  100
/* cop.c */
int
main(void)
{
        int     n, fd1[2], fd2[2];
        pid_t   pid;
        char    line[MAXLINE];

        if (pipe(fd1) < 0 || pipe(fd2) < 0)
                syserr("errore di pipe");

        if ( (pid = fork()) < 0) syserr("fork");
        else if (pid > 0) {   /*processo padre */
                close(fd1[0]);   close(fd2[1]);
                printf("scrivere due numeri: ");
                while (fgets(line, MAXLINE, stdin) != NULL) {
                        n = strlen(line);
                        if (write(fd1[1], line, n) != n)
                                syserr("errore di scrittura in pipe");

                        if ( (n = read(fd2[0], line, MAXLINE)) < 0)
                                syserr("errore di lettura da pipe");

                        printf("somma e prodotto ( dalla pipe ): %s\n",
                                line);
                }
                exit(0);

        } else {          /* processo figlio */
                close(fd1[1]);    close(fd2[0]);
                close(0); dup(fd1[0]);
                close(1); dup(fd2[1]);
                close(fd2[1]);    close(fd1[0]);

                execl("./add", "add", (char *) 0);
                syserr("execl error");
        }
}
```

```
/* CLIFIFO.C */

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/errno.h>


main(int argc,char *argv[])
{
  char buf[100]; /* stringa che invio sulla FIFO */
  int fd; /* descrittore file della FIFO */
  int i;
  int pidpro; /* pid del processo */
  pidpro=getpid();
  /*aggancio la FIFO, aprendola in scrittura:*/
  if((fd=open("ff", O_WRONLY))==-1)
        syserr("Client: open ff");
  printf("Client %d :agganciata la FIFO\n",pidpro);

  /* controllo la presenza dei parametri */
  chkpar(argc,argv,pidpro);

  /*preparo il messaggio:*/
printf("argv = %s  %s\n", argv[0], argv[1]);
  switch(argv[1][0]) {
          case 'c':break;
          case 'm':break;
          case 'r':break;
          case 'v':break;
          case 's':break;
          case 'q':break;
          default : fprintf(stderr,"Client %d (E): comando '%s'
errato\n",
                          pidpro,argv[1]);
                  /* non cancello la FIFO: ci pensa il server */
                    exit(1);
                    break;
  } /*switch*/

  strcpy(buf,argv[1]);
  strcat(buf," ");
  i=2;
  while (argv[i] != NULL)
  {

        strcat(buf,argv[i]);
        strcat(buf," ");
        i++;
  }

  /*invio il comando:*/

  if(write(fd,buf,strlen(buf))==-1)
        syserr("write");
  close(fd);

  printf("Client %d :inviata richiesta (%s)\n",pidpro,argv[1]);

  return(0);

} /*main*/
```

```
/*FUNZIONE:controlla presenza parametri*/
chkpar (argc,argv,pidpro)
  int argc;
  char *argv[];
  int pidpro;
{
  if (argc < 2)
    {
      fprintf (stderr, "Client %d (E): primo argomento assente\n",
                       pidpro);
      /* non cancello la fifo: ci pensa il server */
      exit (1);
    }
  else
    {
      char l1;

      l1 = argv[1][1];
      if ( (l1=='r' || l1=='v' || l1=='s') &&   argc < 3  )
        {
         fprintf (stderr, "Client %d (E): secondo argomento assente\n",
                          pidpro);
         exit (1);
        }
      if ( (l1 == 'm' || l1 == 'c') && argc < 4)
        {
         fprintf (stderr, "Client %d (E): terzo argomento assente\n",
                          pidpro);
         exit (1);
      }
    }

} /* chkpar */
```

```
/* SERFIFO.C */

#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/errno.h>

extern int errno;
main()
{
  int fd; /* descrittore file della FIFO */
  int nread; /* numero byte letti dalla FIFO */
  int i;
  static char buf[100]; /* perche' funziona solo con lo static? */
  char **varg; /* vettore dei comandi letti dalla FIFO */
  char *bufp; /* usato nella strtok */
  struct stat sbuf; /* usata nella chiamata di sistema stat */
  /*extern char **environ;*/


  mknod("ff",S_IFIFO|0666,0);   /* creo la FIFO */
  printf("Server: creata FIFO \"ff\"\n");

  /* apro la FIFO in lettura/scrittura (non in sola lettura altrimenti
      il server si blocca: */
  if((fd=open("ff",O_RDWR))==-1)
        syserr("open ff");
  /*environ[6]=""; PS1=''*/

  /*loop infinito:*/
  while(1)
   {
    printf("\nServer: attendo richieste...\n");
    printf("Comandi:    c file1 file2 (copia)\n");
    printf("            m file1 file2 (move)\n");
    printf("            r file1 (cancella)\n");
    printf("            s file1 (statistica)\n");
    printf("            v file1 (vista di un file)\n");
    printf("            q  (cancella la fifo)\n");
          /* Ricevo il messaggio(se non c'e' nulla attendo) */
    while((nread=read(fd,buf,sizeof(buf))) == 0)
                      ;
    if(nread==-1)
        syserr("read");
    printf("Server: ricevo richiesta {%s}\n",buf);
    bufp=buf;
    for(i=0;i<3;i++) {
         if((varg[i]=strtok(bufp," "))==NULL)
             break;
         bufp=NULL;
    }
```

```
switch(varg[0][0]) {
  case 'c': /*Copia di file*/
            switch(fork()){
            case -1: syserr("fork di cp");
            case 0: printf("Copio \"%s\" in
                    \"%s\"\n",varg[1],varg[2]);
                    execlp("cp","cp",varg[1],varg[2],NULL);
                    syserr("exec di cp");
            default: if(wait(NULL)==-1)
                                      /*il server aspetta la fine*/
                       syserr("wait"); /*dell'operazione di copia*/
            }
            break;
  case 'm': /*Spostamento di file*/
            switch(fork()){
            case -1: syserr("fork di mv");
            case 0: printf("Sposto \"%s\" in
                           \"%s\"\n",varg[1],varg[2]);
                    execlp("mv","mv",varg[1],varg[2],NULL);
                    syserr("exec di mv");
            default: if(wait(NULL)==-1)
                                     /*il server aspetta la fine*/
                       syserr("wait"); /*dello spostamento*/
            }
            break;
  case 'r': /*Cancellazione di file*/
            switch(fork()){
            case -1: syserr("fork di rm");
            case 0: printf("Cancello \"%s\"\n",varg[1]);
                    execlp("rm","rm",varg[1],NULL);
                    syserr("exec di rm");
            default: if(wait(NULL)==-1)
                                 /*il server aspetta la fine*/
                       syserr("wait"); /*della cancellazione*/
            }
            break;
  case 's': /*Statistica di un file*/
            if ( stat(varg[1], &sbuf) < 0 )
               syserr("stat");
            printf("\nStato del file \"%s\" :\n",varg[1]);
            printf("Numero device: %ld\n",sbuf.st_dev);
            printf("Numero inode: %ld\n",sbuf.st_ino);
            printf("Tipo di file e permessi:
                 %#o\n",(int)sbuf.st_mode);
            printf("Numero di hard links: %d\n",(int)
                 sbuf.st_nlink);
            printf("ID utente del proprietario del file: %d\n",
                    sbuf.st_uid);
            printf("ID gruppo del proprietario del file: %d\n",
                    sbuf.st_gid);
            printf("Dimensione in byte del file: %ld\n",
                    (long)sbuf.st_size);
            printf("ultimo accesso: %s",ctime(&(sbuf.st_atime)));
            printf("ultima modifica: %s",ctime(&(sbuf.st_mtime)));
            printf("Ora ultimo cambiamento di stato: %s",
            ctime(&(sbuf.st_ctime)));
            break;
  case 'v': /*Vista di un file*/
            switch(fork()){
            case -1: syserr("fork di cat");
            case 0: printf("Contenuto di \"%s\"\n",varg[1]);
                    execlp("cat","cat",varg[1],NULL);
                    syserr("exec di cat");
            default: if(wait(NULL)==-1)  /*il server aspetta la
```

```
                                                fine*/
                            syserr("wait");  /*dell'operazione di cat*/
                    }
                    break;
          case 'q':  /* cancello la FIFO */
                    close(fd);
                    unlink("ff");
                    printf("Server: FIFO cancellata\n");
                    exit(0);

       } /*switch*/

       for(i=0;i<=30;i++)
            buf[i]='\0';

     }     /* fine del while */

return(0);
} /*main*/
```