- **Errori, errno**

```
/*syserr.c*/
#include <stdio.h>
void syserr(msg)
char *msg;
{
        extern int errno, sys_nerr;
        extern char *sys_errlist[];

        fprintf(stderr,"ERROR: %s (%d",msg,errno);
        if (errno>0&&errno<sys_nerr)
           fprintf(stderr,"; %s)\n",sys_errlist[errno]);
        else
           fprintf(stderr,")\n");
        exit(1);
}



/*fatal.c*/
#include <stdio.h>
void fatal(char *msg)
{
        fprintf(stderr,"ERROR: %s\n", msg);
        exit(1);
}
```

- **Tempi:**

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/times.h>

long times();
#define TICKS 60.

static struct tms tbuf1;
static long real1;

void timestart()
{
        real1 = times(&tbuf1);
}

void timestop(msg)
char *msg;
{
        struct tms tbuf2;
        long real2;

        real2 = times(&tbuf2);
        fprintf(stderr,"%s:real %.2f; usr %.2f; sys %.2f\n",msg,
        (real2-real1)/TICKS,
        (tbuf2.tms_utime-tbuf1.tms_utime)/TICKS,
        (tbuf2.tms_stime-tbuf1.tms_stime)/TICKS);
}
```

```
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */

/*      THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T     */
/*      The copyright notice above does not evidence any        */
/*      actual or intended publication of such source code.     */

#ifndef _SYS_TIMES_H
#define _SYS_TIMES_H

#pragma ident   "@(#)times.h    1.9     92/08/18 SMI"   /* SVr4.0
11.7 */

#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Structure returned by times()
 */
struct tms {
        clock_t tms_utime;              /* user time */
        clock_t tms_stime;              /* system time */
        clock_t tms_cutime;             /* user time, children */
        clock_t tms_cstime;             /* system time, children
*/
};

#if !defined(_KERNEL)
#if defined(__STDC__)
clock_t times(struct tms *);
#else
clock_t times();
#endif
#endif

#ifdef  __cplusplus
}
#endif

#endif  /* _SYS_TIMES_H */
```

- Compilazione, Make

- Esempio di makefile:

```
# questo e' un commento
programma : main.o file1.o file2.o file3.o
        cc –o programma main.o file1.o file2.o file3.o

main.o : main.c
        cc –c main.c

file1.o : file1.c
        cc –c file1.c

file2.o : file2.c
        cc –c file2.c

file3.o : file3.c
        cc –c file3.c
```

- le linee che contengono i due punti specificano le dipendenze. A sinistra dei due punti c'e' il target; a destra ci sono le componenti
- le linee che cominciano con il tab sono linee comando; specificano come costruire i target
- make virifica le date per vedere se i file oggetto sono aggiornati. Se sono aggiornati non fa' niente e scrive semplicemente
  'programma' is up to date
- il diesis rappresenta l'inizio di un commento. Non mettere dopo il tab!
- make si puo' usare con qualsiasi tipo di comandi. Per esempio:

```
report : file1 file2 file
        sort –bdf file1 file2 > /usr/tmp/tempor
        awk –f file /usr/tmp/tempor > report
        rm /usr/tmp/tempor
```

- si possono usare anche caratteri di pattern matching *, ? per identificare i file

- **Uso di file come semafori**

```
#include <sys/errno.h>

#define LOCKDIR "./"
#define MAXTRIES 3
#define NAPTIME 5

int lock(name)
char *name;
{
        char *path, *lockpath();
        int fd, tries;
        extern int errno;
        path = lockpath(name);
        tries = 0;
        while ((fd = creat(path,0)) == -1 && errno ==EACCES) {
           if (++tries >= MAXTRIES)
                return(FALSE);
        }
        if (fd == -1 || close(fd) == -1)
           syserr("lock");
        return(TRUE);
}

void unlock(name)
char *name;
{
        char *lockpath();

        if (unlink(lockpath(name)) == -1)
           syserr("unlock");
}

static char *lockpath(name)
char *name;
{
        char *nn;

        static char path[20];
        char *strcat();

        strcpy(path, LOCKDIR);
        nn = strcat(path,name);
        printf("lockpath: %s\n",nn);

        return(nn);
}
```

- **Copia di files: 1° tentativo (copy.c)**

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#define BUFSIZE 512
void copy(char *from, char *to);

main()
{
   char nome1[10], nome2[10];
   printf("nomi file? "); scanf("%s %s",nome1,nome2);
   copy(nome1,nome2);
}

void copy(from,to)
char *from, *to;
{
        int fromfd, tofd, nread;
        char buf[BUFSIZE];

        if((fromfd = open(from,O_RDONLY,0666)) == -1)     syserr(from);
        if ((tofd = creat(to,0666)) == -1)     syserr(to);
        while ((nread = read(fromfd, buf, sizeof(buf))) > 0)
           if(write(tofd, buf, nread) != nread)
                syserr("write");
        if(nread == -1)   syserr("read");
        if(close(fromfd) == -1 || close(tofd) == -1) syserr("close");
}
```

- **Copia files: tiene conto delle scritture incomplete (copyok.c)**

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#define BUFSIZE 512
void copyok(char *from, char *to);
main()
{
   char nome1[10], nome2[10];
   printf("da ... a... "); scanf("%s %s",nome1,nome2);
   copyok(nome1,nome2);
}

void copyok(char *from, char *to)
{
        int fromfd, tofd, nread, nwrite, n;
        char buf[BUFSIZE];

        if((fromfd = open(from,0)) == -1)       syserr("from");
        if ((tofd = creat(to,0666)) == -1)      syserr("to");
        while ((nread = read(fromfd, buf, sizeof(buf))) != 0) {
           if (nread == -1)   syserr("read");
           nwrite = 0;
           do{
                if ((n=write(tofd,&buf[nwrite], nread - nwrite)) == -1)
                        syserr("write");
                nwrite += n;
           } while (nwrite < nread);
        }
        if(close(fromfd) == -1 || close(tofd) == -1) syserr("close");
}
```

- **User buffering (Stream.c copybuffered.c)**

```
/* stream.h */
#define SENOMEM 1001
#define SEINVAL 1002
#define BUFSIZE 512

typedef struct{
        int fd;
        char dir;
        int total;
        int next;
        char buf[BUFSIZE];
} STREAM;

/*stream.c */
#include <fcntl.h>
#include <stdio.h>
#include "boolean.h"
#include "stream.h"

extern int errno;

STREAM *Sopen(char *path,char *dir)
{
        STREAM *z;
        int fd, flags;
        char *malloc();
        switch(dir[0]){
        case 'r':
                flags = O_RDONLY;
                break;
        case 'w':
                flags = O_WRONLY | O_CREAT | O_TRUNC;
                break;
        default:
                errno = SEINVAL;
                return(NULL);
        }
        if((fd=open(path,flags,0666)) == -1)
                return(NULL);
        if((z = (STREAM *)malloc(sizeof(STREAM))) == NULL) {
                errno = SENOMEM;
                return(NULL);
        }
        z->fd = fd;
        z->dir = dir[0];
        z->total = z->next = 0;
        return(z);
}

static BOOLEAN readbuf(STREAM *z)
{
        switch(z->total = read(z->fd, z->buf, sizeof(z->buf))) {
        case -1:
                return(FALSE);
        case 0:
                errno = 0;
                return(FALSE);
        default:
                z->next = 0;
                return(TRUE);
        }
}
```

```
static BOOLEAN writebuf(STREAM *z)
{
        int n,total;

        total=0;
        while(total < z->next){
                if((n=write(z->fd,&z->buf[total],z->next - total))==-1)
                        return(FALSE);
                total += n;
        }
        z->next = 0;
        return(TRUE);
}
int Sgetc(STREAM *z)
{
        int c;

        if(z->next >= z->total && !readbuf(z))
                return(-1);
        return(z->buf[z->next++] & 0377);
}
BOOLEAN Sputc(STREAM *z,char c)
{
        z->buf[z->next++] = c;
        if (z->next >= sizeof(z->buf))
                return(writebuf(z));
        return(TRUE);
}
BOOLEAN Sclose(STREAM *z)
{
        int fd;

        if(z->dir == 'w' && !writebuf(z))
                return(FALSE);
        fd = z->fd;
        free(z);
        return(close(fd) != -1);
}
/*************************************/
#include "boolean.h"
#include "stream.h"
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
void copybuffered(char *from, char *to);
void timestart();
void timestop(char *msg);

main()
{
   char nome1[10], nome2[10];
   printf("nomi file? "); scanf("%s %s",nome1,nome2);
   copybuffered(nome1,nome2);
}
void copybuffered(char *from, char *to)
{
        STREAM *stfrom, *stto;
        int c;    extern int errno;
        if((stfrom = Sopen(from,"r")) == NULL) syserr(from);
        if((stto = Sopen(to,"w")) == NULL)  syserr(to);
        while((c=Sgetc(stfrom)) != -1)
                if(!Sputc(stto,c)) syserr("Sputc");
        if(errno != 0) syserr("Sgetc");
        if (!Sclose(stfrom) || !Sclose(stto))   syserr("Sclose");
}
```

- **Crea un file con una interruzione (ls2.c, verificare con od –c <file>)**

```
#include        <sys/types.h>
#include        <sys/stat.h>
#include        <fcntl.h>
#include        "head.h"
char    buf1[] = "abcdefghij"; char    buf2[] = "ABCDEFGHIJ";
int main(void)
{
    int             fd;
    if ( (fd = creat("file.hole", FILE_MODE)) < 0)  syserr(" errore in creat");
    if (write(fd, buf1, 10) != 10)          syserr("errore di scrittura");
    if (lseek(fd, 40, SEEK_SET) == -1)      syserr("errore di lseek");
    if (write(fd, buf2, 10) != 10)          syserr("errore di scrittura");
        exit(0);
}
```

- **Lettura all'indietro di un file. (back.c)**

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
void syserr(char *msg);
void backward(char *path);
void fatal(char *msg);
main()
{
   char nome[10];
   printf("nome del file di ingresso? "); scanf("%s",nome);
   backward(nome);
}
void backward(char *path)
{
   char s[101], c;
   int i, fd, nread;
   long where, lseek();
   if ((fd=open(path,0)) == -1)
      syserr(path);
   if((where=lseek(fd,-1L,2))==-1)
      syserr("lseek");
   i=sizeof(s)-1;
   s[i]='\0';
   while ((nread=read(fd,&c,1)) == 1){
      if(c=='\n'){
        printf("%s",&s[i]);
        i=sizeof(s)-1;
      }
      if (i==0)
        fatal("line too long");
      s[--i] = c;
      if(where == 0)
        break;
      if((where = lseek(fd,-2L,1)) == -1)
        syserr("lseek");
    }
    switch(nread) {
    case 0:
        fatal("end-of-file");
    case -1:
        syserr("read");
    }
    printf("%s",&s[i]);
    if(close(fd) == -1)
        syserr("close");
}
```

- **Chiamata di sistema stat(). Programma statutil.c**

```
#include <stdio.h>
#include "boolean.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>

void mainloop();
void help();
void prompt(char *msg, char *result);
void status(char *path);
void dspstatus(struct stat *sbp);
long timecvt(char *atime);
void chtime(char *path, char which);
void chowner(char *path);
void chperms(char *path);
void syserrmsg(char *msg);

main()
{
      help();
      mainloop();
}
static void mainloop()  /* elabora i comandi */
{
   char path[50], cmd[10], shcmd[100];
   while(1) {
        prompt("Comando",cmd);
        if(strlen(cmd) > 1)
           cmd[0] = '\1'; /* forza il messaggio 'comando sconosciuto' */
        switch(cmd[0]){
        case '\0':
        case 'q':
           exit(0);
        case 'a':
        case 'm':
           chtime(path,cmd[0]);       continue;
        case 'f':
           prompt("File",path);
           if (access(path,0) == -1){
                printf("%s non esiste\n",path);      continue;
           }
           status(path);       continue;
        case 'o':
           chowner(path);    continue;
        case 'p':
           chperms(path);      continue;
        case 's':
           status(path);          continue;
        case '!':
           prompt("Comando di Shell",shcmd);
           system(shcmd);      continue;
        case '?':
           help();            continue;
        default:
           printf("Comando sconosciuto; usa ? per help");
           continue;
        }
     }
}
```

```
static void help()
{
   printf("a    cambia il tempo di accesso\n");
   printf("f    nuovo nome file\n");
   printf("m    cambia il tempo di modifica\n");
   printf("o    cambia il proprietario\n");
   printf("p    cambia i permessi\n");
   printf("q    fine\n");
   printf("s    mostra lo stato\n");
   printf("!    esegue comandi UNIX\n");
   printf("?    mostra questo menu\n");
}

static void prompt(char *msg, char *result)
{
   printf("\n%s ? ",msg);
   if(gets(result) == NULL)   exit(0);
}

static void status(char *path)  /* comando 's' */
{
    struct stat sb;

    if (stat(path,&sb) == -1){
       syserrmsg("stat");
       return;
    }
    printf("\nFile \"%s\"\n",path);
    dspstatus(&sb);
}
```

```
static void dspstatus(struct stat *sbp)  /* mostra lo stato */
{
    BOOLEAN isdevice = FALSE;
    struct passwd *pw, *getpwuid();
    struct group *gr, *getgrgid();
    char *name, *asctime();
    struct tm *localtime();
    if ((sbp->st_mode & S_IFMT) == S_IFDIR)
       printf("Direttorio\n");
    else if ((sbp->st_mode & S_IFMT) == S_IFBLK){
       printf("File speciale a blocchi\n");
       isdevice = TRUE;
    } else if ((sbp->st_mode & S_IFMT) == S_IFCHR) {
       printf("File speciale a caratteri\n");
       isdevice = TRUE;
    } else if ((sbp->st_mode & S_IFMT) == S_IFREG)
       printf("File ordinario\n");
    else if ((sbp->st_mode & S_IFMT) == S_IFIFO)
       printf("FIFO\n");
    if (isdevice)
       printf("Numero del dispositivo: %d, %d\n", (sbp->st_rdev >> 8) & 0377,
          sbp->st_rdev & 0377);
    printf("Risiede su : %d, %d\n",(sbp->st_dev >> 8) & 0377,
          sbp->st_dev & 0377);
    printf("I-node: %d; Links: %d, Dimensione: %ld\n", sbp->st_ino,
          sbp->st_nlink, sbp->st_size);
    if(( pw = getpwuid(sbp->st_uid)) == NULL)
          name = "???";
    else
          name = pw->pw_name;
    printf("ID del proprietario: %d; Nome: %s\n", sbp->st_uid, name);
    if((gr = getgrgid(sbp->st_gid)) == NULL)
         name = "???";
    else
         name = gr->gr_name;
    printf("ID del gruppo: %d; Nome: %s\n", sbp->st_gid,name);
    if((sbp->st_mode & S_ISUID) == S_ISUID)
        printf("Set-user-ID\n");
    if((sbp->st_mode & S_ISGID) == S_ISGID)
        printf("Set-group-ID\n");
    if((sbp->st_mode & S_ISVTX) == S_ISVTX)
        printf("Salva testo swappato dopo uso\n");
    printf("Permessi: %o\n",sbp->st_mode & 0777);
    printf("Ultimo accesso:     %s",
        asctime(localtime(&sbp->st_atime)));
    printf("Ultima modifica:    %s",
        asctime(localtime(&sbp->st_mtime)));
    printf("Ultimo cambiamento di stato:     %s",
        asctime(localtime(&sbp->st_ctime)));
}
```

```
static void chtime(char *path, char which) /* comandi 'a' e 'm' */
{
   char atime[20];
   long seconds/* ,timecvt()*/ ;
   struct stat sb;
   struct utimbuf{
        time_t actime;
        time_t modtime;
   } tb;

   if (stat(path, &sb) == -1) {
        syserrmsg("stat");     return;
   }
      prompt("Time (YYMMDDhhmmss)", atime);
   if((seconds = timecvt(atime)) <= 0)    return;
    switch(which){
   case 'a':
        tb.actime = seconds;
        tb.modtime = sb.st_mtime;
        break;
   case 'm':
        tb.actime = sb.st_atime;
        tb.modtime = seconds;
   }
   if(utime(path,&tb) == -1)
        syserrmsg("utime");
}

static void chowner(char *path) /* comando 'o' */
{
        char oname[20], gname[20];
        int owner, group;
        struct passwd *pw, *getpwnam();
        struct group *gr, *getgrnam();

        prompt("Nome del proprietario", oname);
        if((pw = getpwnam(oname)) == NULL){
           printf("Proprietario sconosciuto\n");
           return;
        }
        owner = pw->pw_uid;
        prompt("Nome del gruppo",gname);
        if((gr = getgrnam(gname)) == NULL) {
                printf("Gruppo sconosciuto\n");
                return;
        }
        group = gr->gr_gid;
        if(chown(path, owner, group) == -1)
           syserrmsg("chown");
}
```

```
static void chperms(char *path) /* comando 'p' */
{
    char smode[20];
    int mode;

    prompt("Modo (fino a 4 cifre ottali)", smode);
    if(sscanf(smode, "%o", &mode) != 1){
       printf("Modo non valido\n");
       return;
    }
    if(chmod(path, mode) == -1)
       syserrmsg("chmod");
}

static void syserrmsg(char *msg) /* visualizza messaggi d'errore */
{
   extern int errno, sys_nerr;
   extern char *sys_errlist[];

   fprintf(stderr,"ERROR: %s (%d", msg, errno);
   if(errno > 0 && errno < sys_nerr)
        fprintf(stderr,"; %s)\n", sys_errlist[errno]);
   else
        fprintf(stderr,")\n");
}
```

- **Questo programma verifica il modo di apertura di un file (fc1.c)**

```
#include        <sys/types.h>
#include        <fcntl.h>
#include        "head.h"

int main(int argc, char *argv[])
{
        int             accmode, val,fd;

        fd=open("file.hole",O_RDWR);
        if ( (val = fcntl(fd, F_GETFL, 0)) < 0)
                syserr("errore di fcntl ");

        accmode = val & O_ACCMODE;
        if      (accmode == O_RDONLY)   printf("sola lettura");
        else if (accmode == O_WRONLY)   printf("sola scrittura");
        else if (accmode == O_RDWR)             printf("lettura e scrittura");
        else syserr("modo accesso sconosciuto");

        putchar('\n');
        exit(0);
}
```

Questo programma (ck1.c) verifica il modo d'accesso ad un file

```
#include        <sys/types.h>
#include        <fcntl.h>
#include        "head.h"

int
main(int argc, char *argv[])
{
        if (argc != 2)
                syserr("Uso: a.out <path>");
/*il pattern di access e' l'OR di R_OK, W_OK, X_OK, F_OK(esistenza)*/
        if (access(argv[1], R_OK) < 0)
                syserr("errore di access per %s", argv[1]);
        else
                printf("access in lettura\n");

        if (open(argv[1], O_RDONLY) < 0)
                syserr("errore di apertura per %s", argv[1]);
        else
                printf("errore di apertura\n");

        exit(0);
}
```

- **Questo programma (ck3.c) tronca un file mantenendo inalterati i tempi**

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

int
main(int argc, char *argv[])
{
        int             i;
        struct stat     statbuf;
        struct utimbuf  timebuf;

        for (i = 1; i < argc; i++) {
                if (stat(argv[i], &statbuf) < 0) {      /* preleva i tempi
correnti */
                        syserr("%s: errore di stat", argv[i]);
                        continue;
                }
                if (open(argv[i], O_RDWR | O_TRUNC) < 0) {      /* tronca */
                        syserr("%s: errore di apertura", argv[i]);
                        continue;
                }
                timebuf.actime  = statbuf.st_atime;
                timebuf.modtime = statbuf.st_mtime;
                if (utime(argv[i], &timebuf) < 0) {             /* azzera i
tempi */
                        syserr("%s: errore di utime", argv[i]);
                        continue;
                }
        }
        exit(0);
}
```