```
/* envlib.c */
#include <stdio.h>
#include "boolean.h"

#define MAXVAR 25

static struct ambiente{
        char *name;             /* nome variabile */
        char *val;              /* valore variabile */
        BOOLEAN exported;        /* deve essere esportata ? */
} tab[MAXVAR];

extern char **environ;

static struct ambiente *find(name)
char *name;
{
        int i;
        struct ambiente *v;

        v = NULL;
        for (i=0; i<MAXVAR; i++)
            if(tab[i].name == NULL){
                if(v == NULL)
                   v = &tab[i];
            }
            else if (strcmp(tab[i].name,name) == 0){
                v = &tab[i];
                break;
            }
        return(v);
}

char *malloc(), *realloc();
static BOOLEAN assign(p,s)
char **p, *s;
{
        int size;

        size = strlen(s)+1;
        if(*p == NULL){
           if ((*p = malloc(size)) == NULL)
               return(FALSE);
        }
        else if ((*p = realloc(*p, size)) == NULL)
                return(FALSE);
        strcpy(*p,s);
        return(TRUE);
}

BOOLEAN EVset(name,val)
char *name, *val;
{
        struct ambiente *v;
        if((v = find(name)) == NULL)
           return(FALSE);
        return(assign(&v->name,name) && assign(&v->val,val));
}

BOOLEAN EVexport(name)
char *name;
{
        struct ambiente *v;
        if ((v = find(name)) == NULL)
                return(FALSE);
        if(v->name == NULL)
                if(!assign(&v->name, name) || !assign(&v->val, ""))
                        return(FALSE);
        v->exported = TRUE;
        return(TRUE);
}
```

```
char *EVget(name)
char *name;
{
        struct ambiente *v;

        if((v = find(name)) == NULL || v->name == NULL)
                return(NULL);
        return(v->val);
}


BOOLEAN EVinit()
{
        int i, namelen;
        char name[20];

        for(i=0; environ[i] != NULL; i++){
                namelen = strcspn(environ[i], "=");
                strncpy(name, environ[i], namelen);
                name[namelen] = '\0';
                if(!EVset(name, &environ[i][namelen + 1]) || !EVexport(name))
                        return(FALSE);
        }
        return(TRUE);
}


BOOLEAN EVupdate()
{
        int i,envi,nvlen;
        struct ambiente *v;
        static BOOLEAN updated = FALSE;

        if(!updated)
                if((environ = (char**)malloc((MAXVAR +1) * sizeof(char*)))
                == NULL)
                  return(FALSE);
        envi = 0;
        for(i=0; i<MAXVAR; i++){
                v = &tab[i];
                if(v->name == NULL || !v->exported)
                        continue;
                nvlen = strlen(v->name) + strlen(v->val) +2;
                if(!updated){
                        if((environ[envi] = malloc(nvlen)) == NULL)
                           return(FALSE);
                }
                else if((environ[envi] = realloc(environ[envi], nvlen)) == NULL)
                   return(FALSE);
                sprintf(environ[envi], "%s=%s", v->name, v->val);
                envi++;
         }
         environ[envi] = NULL;
         updated = TRUE;
         return(TRUE);
}

void EVprint()
{
        int i;

        for (i=0;i<MAXVAR;i++)
            if(tab[i].name!= NULL)
               printf("%3s %s=%s\n", tab[i].exported ? "[E]":"",
                 tab[i].name, tab[i].val);
}
```

```
/*********************************************/
/*    testenv.c                              */
/*********************************************/

#include "boolean.h"
#include <stdio.h>

main()
{
        if(!EVinit())
          fatal("non posso inizializzare l'ambiente");
        printf("Prima dell'aggiornamento:\n");
        EVprint();
        if(!EVset("contatore","0"))
          fatal("EVset");
        if(!EVset("processo","/home/mumolo/hello") || !EVexport("processo"))
          fatal("EVset oppure EVexport");
        printf("\nDopo l'aggiornamento:\n");
        EVprint();
        exit(0);
}
```

```
/**************/
/* prstatus.c */
/**************/
#include        <sys/types.h>
#include        <sys/wait.h>
#include        "head.h"

void
pr_exit(int status)
{
        if (WIFEXITED(status))
                printf("terminazione normale, stato = %d\n",
                                WEXITSTATUS(status));
        else if (WIFSIGNALED(status))
                printf("terminazione anormale, signal number = %d%s\n",
                                WTERMSIG(status),
                                "");
        else if (WIFSTOPPED(status))
                printf("processo figlio fermato, signal = %d\n",
                                WSTOPSIG(status));
}


/*******************/
/*    runchild.c   */
/*******************/

#include        <sys/types.h>
#include        <sys/wait.h>

int
main(void)
{
        pid_t   pid;
        int     status;

        if ( (pid = fork()) < 0)
                syserr("errore in fork");
        else if (pid == 0)              /* processo figlio */
                exit(7);

        if (wait(&status) != pid)       /*  attesa */
                syserr(" errore in wait");
        pr_exit(status);                /* stampa lo stato */

        if ( (pid = fork()) < 0)
                syserr("errore in fork");
        else if (pid == 0)              /* processo figlio */
                abort();                /* genera SIGABRT */

        if (wait(&status) != pid)       /* attesa */
                syserr("errore in wait");
        pr_exit(status);                /* stampa lo stato */

        if ( (pid = fork()) < 0)
                syserr("errore in fork ");
        else if (pid == 0)              /* processo figlio */
                status /= 0;            /* genera SIGFPE */

        if (wait(&status) != pid)       /* attesa */
                syserr(" errore in wait");
        pr_exit(status);               /* stampa lo stato */

        exit(0);
}
```

```
/******************/
/*    execex.c   */
/******************/
#include        <sys/types.h>
#include        <sys/wait.h>

char    *env_init[] = { "USER=unknown", "PATH=/tmp", NULL };
int
main(void)
{
        pid_t   pid;

        if ( (pid = fork()) < 0)
                syserr(" errore di fork");
        else if (pid == 0) {    /* specifica il path, specifica l'ambiente */
                if (execle("/usr/bin/echo",
                                "echoall", "myarg1", "MY ARG2", (char *) 0,
                                env_init) < 0)
                        syserr("errore di execle");
        }
        if (waitpid(pid, NULL, 0) < 0)
                syserr("errore di wait");

        if ( (pid = fork()) < 0)
                syserr(" errore di fork");
        else if (pid == 0) {    /* specifica il nome file  e eredita ambiente */
                if (execlp("echo",
                                "echoall", "only 1 arg", (char *) 0) < 0)
                        syserr("errore di execlp ");
        }
        exit(0);
}

/******************/
/*    systemex.c   */
/******************/
#include        <sys/types.h>
#include        <sys/wait.h>
#include        <errno.h>
#include        <unistd.h>
int
system(const char *cmdstring)   /* questa versione non gestisce segnali */
{
        pid_t   pid;
        int             status;

        if (cmdstring == NULL)
                return(1);

        if ( (pid = fork()) < 0) {
                status = -1;    /* probabimente fine processi */

        } else if (pid == 0) {  /* processo figlio */
                execl("/bin/sh", "sh", "-c", cmdstring, (char *) 0);
                _exit(127);     /* errore di execl  */

        } else {                /* processo padre */
                while (waitpid(pid, &status, 0) < 0)
                        if (errno != EINTR) {
                                status = -1; /* errore di waitpid() */
                                break;
                        }
        }

        return(status);
}
```

```
/******************/
/*    systemex1.c   */
/******************/

#include        <sys/types.h>
#include        <sys/wait.h>

int
main(void)
{
        int             status;

        if ( (status = system("date")) < 0)
                syserr("errore system() ");
        pr_exit(status);

        if ( (status = system("nosuchcommand")) < 0)
                syserr("errore system()");
        pr_exit(status);

        if ( (status = system("who; exit 44")) < 0)
                syserr("errore system() ");
        pr_exit(status);

        exit(0);
}



/******************/
/*    systemex2.c   */
/******************/
int
main(int argc, char *argv[])
{
        int             status;

        if (argc < 2)
                fatal("argomenti in linea richiesti");

        if ( (status = system(argv[1])) < 0)
                syserr("errore system() ");
        pr_exit(status);

        exit(0);
}
```