EJEMPLOS DE UTILIZACIÓN DE LAS LLAMADAS AL SISTEMA DEL SUBSISTEMA DE FICHEROS EN UNIX

```
#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <io.h>

#include <string.h>

int main(void)

{

int handle;

char string[40];

int length, res;

/* Crea un fichero llamado "TEST.$$$" en el directorio de trabajo y escribe en Jl una cadena. Si "TEST.$$$" existe ya, ser< sobreescrito. */

if ((handle = open("/users/TEST.$$$", O_CREAT | O_TRUNC,0700))== −1)

{

printf("open");

exit(1);

}

strcpy(string, "Hello, world!\n");

length = strlen(string);

if ((res = write(handle, string, length)) != length)

{

printf("write");

exit(1);
```

```
}
printf("Escritos %d bytes en el archivo.\n", res);
close(handle);
return 0;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Lectura y escritura de ficheros */
#include <stdio.h>
#include <io.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#define TAM_BUF 128
void entrada(char *buf, int fd1);
void mostrar(char *buf, int fd2);
void main(void)
{
char buf(TAM_BUF);
int fd1, fd2;
if((fd1=open("/users/m1306854/prueba", O_WRONLY))==−1){
printf("No se puede abrir el archivo\n");
exit(1);
}
entrada(buf, fd1);
close(fd1);
if((fd2=open("prueba", O_RDONLY))==−1){
```

```
printf("No se puede abrir el archivo\n");
exit(1);
}
mostrar(buf, fd2);
close(fd2);
}
void entrada(char *buf, int fd1)
{
register int t;
do{
for(t=0; t<TAM_BUF; t++)
buf[t]='\0';
gets(buf);
if(write(fd1, buf, TAM_BUF) != TAM_BUF){
printf("Error de escritura en el archivo\n");
exit(1);
}
}while (strcmp(buf, "salir"));
}
void mostrar(char *buf, int fd2)
{
for(;;){
if(read(fd2, buf, TAM_BUF)==0)
return;
printf("%s\n", buf);
}
```

```
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Lectura de un directorio */
#include<sys/dir.h>
#include<stdio.h>
int directorio(path);
char *path;
{
struct directx{
ino_t d_ino;
char d_name[DIRSIZ +1];
}dlink;
int fd,nread;
char dname[DIRSIZ +1];
if ((fd=open(path,0))== −1)
{ perror(path);
exit(1); }
dlink.d_name[DIRSIZ+1]='\0';
while ((nread=read(fd,&dlink,sizeof(struct directx)))==sizeof(struct directx))){
if (dlink.d_ino==0)
dname="−−−−−−−−−−−sin uso−−−−−−−−−−";
else
strcpy (dname,dlink.d_name);
printf("%−14s%4d\n",dname,dlink.d_ino);
}
}
```

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Creaci\n de un directorio con mknod */

#define ERROR −1

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

main(int argc, char **argv)

{ if (argc!=2)

{ printf("((( Ndmero de par<metros incorrecto !!!\n");

return 1; }

if (mknod(argv[1],S_IFDIR | 0755,0)==ERROR)

{ errorSistema(argv[1]);

return(2); }

return 0;

}

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Ejemplo de la llamada al sistema link */

#define ERROR −1

#include <stdio.h>

main(int argc, char **argv)

{ if (argc!=3)

{ printf("((( Ndmero de par<metros incorrecto !!!\n");

return 1; }

if (link(argv[1],argv[2])==ERROR)

{ errorsistema("link");

return(2); }
```

```
return 0;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema chown */
#define ERROR −1
#include <stdio.h>
main(int argc, char **argv)
{ int uid,gid;
if (argc!=4)
{ printf("((( Ndmero de par<metros incorrecto !!!\n");
return 1; }
uid=atoi(argv[1]);
gid=atoi(argv[2]);
if (chown(argv[3],uid,gid)==ERROR)
{ errorSistema("chown");
return(3); }
return 0;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema utime */
#define ERROR −1
#include <stdio.h>
#include <sys/types.h>
struct utimbuf {
time_t acceso_time;
time_t modifi_time;
```

```
} *times;
main(int argc, char **argv)
{ long int ahora;
if (argc!=2)
{ printf("((( Ndmero de par<metros incorrecto !!!\n");
return 1; }
if ((ahora=time(0))==ERROR)
{ errorSistema("time");
return(2); }
times=(struct utimbuf*)malloc(sizeof(struct utimbuf));
times−>acceso_time=ahora;
times−>modifi_time=ahora;
if(utime(argv[1],times)==ERROR)
{ errorSistema ("utime");
return(3); }
return 0;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema stat */
#define ERROR −1
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
main(int argc, char **argv)
{ struct stat buffer;
if (argc!=2)
```

```
{ printf("((( Ndmero de par<metros incorrecto !!!\n");
return 1; }
if (stat(argv[1],&buffer)==ERROR)
{ errorSistema("stat");
return(2); }
printf("Datos del i−nodo del fichero : %s−1n",argv[1]);
printf("============================\n");
printf("Ndmero del i−nodo = %d \n",buffer.st_ino);
printf("Tipo de fichero = ");
switch(buffer.st_mode & S_IFMT)
{ case S_IFDIR : printf("Directorio \n"); break;
case S_IFREG : printf("Fichero regular \n"); break;
case S_IFCHR : printf("Dispositivo de tipo car<cter \n"); break;
case S_IFBLK : printf("Dispositivo de tipo bloque \n"); break;
case S_IFIFO : printf("TuberRa nominada \n"); break;
default : printf("tipo desconocido\n"); }
printf("Sus permisos son = ");
/* Permisos para el propietario */
switch(buffer.st_mode & 0000700) {
case 0700 : printf("rwx "); break;
case 0600 : printf("rw− "); break;
case 0500 : printf("r−x "); break;
case 0400 : printf("r−− "); break;
case 0300 : printf("−wx "); break;
case 0200 : printf("−w− "); break;
case 0100 : printf("−−x "); break;
```

```
case 0000 : printf("−−− "); break; }
/* Permisos para el grupo */
switch(buffer.st_mode & 0000070) {
case 0070 : printf("rwx "); break;
case 0060 : printf("rw− "); break;
case 0050 : printf("r−x "); break;
case 0040 : printf("r−− "); break;
case 0030 : printf("−wx "); break;
case 0020 : printf("−w− "); break;
case 0010 : printf("−−x "); break;
case 0000 : printf("−−− "); break; }
/* Permisos para los dem<s */
switch(buffer.st_mode & 0000007) {
case 0007 : printf("rwx "); break;
case 0006 : printf("rw− "); break;
case 0005 : printf("r−x "); break;
case 0004 : printf("r−− "); break;
case 0003 : printf("−wx "); break;
case 0002 : printf("−w− "); break;
case 0001 : printf("−−x "); break;
case 0000 : printf("−−− "); break; }
printf("Ndmero de links = %d\n", buffer.st_nlink);
printf("Propietario UID = %d\n",buffer.st_uid);
printf("Grupo gid = %d\n", buffer.st_gid);
printf("Ndmero de Device = %d\n",buffer.st_rdev);
printf("TamaZo en bytes = %d\n", buffer.st_size);
```

```
printf("Valor de st_atime= %s\n",ctime(&buffer.st_atime));
printf("Valor de st_mtime= %s\n",ctime(&buffer.st_mtime));
printf("Valor de st_ctime= %s\n",ctime(&buffer.st_ctime));
return 0;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema ustat */
#define ERROR −1
#include <stdio.h>
#include <sys/types.h>
#include <ustat.h>
main(int argc, char **argv)
{ int numero;
struct stat buffer;
if (argc!=2)
{ printf("((( Ndmero de par<metros incorrecto !!!\n");
return 1; }
numero=atoi(argv[1]);
if (ustat(numero,&buffer)==ERROR)
{ errorSistema("ustat");
return(2); }
printf("Datos del device ndmero : %s\n",argv[1]);
printf("============================\n");
printf("Total de bloques libres = %d \n",buffer.f_tfree);
printf("Total de i−nodos libres = %d\n", buffer.f_tinode);
printf("Nombre del File System = %d\n",buffer.f_fname);
```

```
printf("Nombre del directorio \n");
printf(" sobre el que se monta= %s\n",buffer.f_fpack);
return 0;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema dup */
#define ERROR −1
#include <stdio.h>
#include <fcntl.h>
main(int argc, char **argv)
{ long int fd1,fd2;
if ((fd1=open("DATOS",O_RDONLY))==ERROR)
{ errorsistema ("open");
return 1; }
if ((fd2=dup(fd1))==ERROR)
{ errorsistema("dup");
return(2); }
printf("Valor del primer descriptor de archivo : %ld\n",fd1);
printf("Valor del segundo descriptor de archivo: %ld\n",fd2);
close(fd1);
close(fd2);
return 0;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema fstat */
#include <stdio.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#define STDIN 0
#define ERROR −1
int main(argc, argv)
int argc;
char *argv[];
{
int rc=0, i;
struct stat statb;
if(argc==1)
if(fstat(STDIN, &statb)==ERROR){
perror("stdin");
rc=1;
} else presenta("stdin", &statb);
else
for(i=1; i<argc; i++)
if(stat(argv[i], &statb)==ERROR){
perror(argv[i]);
rc=1;
} else presenta(argv[i], &statb);
exit(rc);
}
int presenta(nombre, sp)
char nombre;
struct stat *sp;
```

```
{
extern char *ctime();
printf("Fichero: %s\n", nombre);
printf("Ndmero de nodo−i: %d\n", sp−>st_ino);
printf("Modo: %o\n", cambia(sp−>st_mode));
printf("Enlaces: %d\n", sp−>st_nlink);
printf("ID del usuario: %d\n", sp−>st_uid);
printf("ID de grupo: %d\n", sp−>st_gid);
printf("TamaZo: %ld\n", sp−>st_size);
printf("Ultimo acceso: %s\n", ctime(&sp−>st_atime));
printf("Ultima modificaci\n: %s\n", ctime(&sp−>st_mtime));
printf("Creaci\n: %s\n", ctime(&sp−>st_ctime));
}
char cambia(modo)
int modo;
{
if((modo & 512)==1)
return('r');
}
−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema access */
#include <stdio.h> /* cabecera de e/s estandar */
#include <sys/access.h> /* accesibilidad de los archivos */
main(int argc,char *argv[])
{
if (argc==1)
```

```
printf("ERROR: debe especificarse un archivo \n"),
exit(1);
if (access(argv[1],00)!=0)
printf("%s: no existe \n",argv[1]);
if (access(argv[1],04)==0)
printf("%s: Tiene permiso de lectura \n",argv[1]);
if (access(argv[1],02)==0)
printf("%s: Tiene permiso de escritura \n",argv[1]);
if (access(argv[1],01)==0)
printf("%s: Tiene permiso de ejecucion \n",argv[1]);
}
```

```
d#include <stdio.h> /* cabecera de e/s estandar */
#include <fcntl.h> /* atributos de ficheros */
#include <sys/mode.h> /* valores de modo de ficheros */
#include <sys/types.h> /* definiciones de tipos */
#include <unistd.h> /* constantes de lseek */
main()
{
int fd;
int fd1;
int ret;
int retc;
short i;
char line[31];
retc=chdir("/home/domingo");
```

```
/* abre el fichero mifichero para lectura y escritura */
if ((fd=open("michero",O_RDWR,0644))!=0) {
/* crea fichero nuevofich con acceso de lectura al grupo */
retc=umask(S_IROTH);
if ((fd1=creat("nuevofich",0644))!=0) {
/* salta 5 primeras entradas de michero */
ret=lseek(fd,(5*sizeof(line)),SEEK_SET);
/* lee 5 siguientes entradas de michero
y las escribe en nuevofich */
for (i=0;i<5;i++) {
retc=read(fd,line,sizeof(line));
retc=write(fd1,line,sizeof(line));
}
close(fd1);
}
close(fd);
chmod("mifichero",S_IRUSR|S_IWUSR|S_IRGRP|S_IWGRP);
}
else
perror ("fichero mifichero inexistente");
}
```

```
/* Establecimiento de bloqueos a ficheros */
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
```

```
#include <unistd.h>
#ifdef SEEK_SET
# define SEEK_SET 0
#endif
#define EQ(str1,str2) (strcmp(str1,str2)==0)
#define FICHERO "tmp"
sin_bloqueo()
{}
con_bloqueo(fd,orden)
int fd, orden;
{
struct flock cerrojo;
cerrojo.l_type=orden;
cerrojo.l_whence=SEEK_SET;
cerrojo.l_start=0;
cerrojo.l_len=0;
if (fcntl(fd,F_SETLKW,&cerrojo)==−1)
{
perror("fcntl");
exit (−1);
}
}
main(argc,argv)
int argc;
char *argv[];
{
```

```
int fd;
int numero,i,j;
int (*bloquear)();
if (argc==1)
bloquear=sin_bloqueo;
else if (argc==2 && EQ(argv[1],"−b"))
bloquear=con_bloqueo;
else{
fprintf (stderr,"Forma de uso: %s [−b]\n",argv[0]);
exit (−1);
}
if ((fd=open(FICHERO,O_RDWR|O_CREAT,0644))==−1)
{
perror(FICHERO);
exit (1);
}
if (read(fd,&numero,sizeof(numero))!=sizeof(numero))
{
numero=0;
write (fd,&numero,sizeof(numero));
}
for (i=0;i<10;i++){
lseek(fd,0L,SEEK_SET);
(*bloquear)(fd,F_WRLCK);
read (fd,&numero,sizeof(numero));
++numero;
```

```
sleep(1);
lseek(fd,0L,SEEK_SET);
write (fd,&numero,sizeof(numero));
printf("PID=%d, nro =%d\n",getpid(),numero);
(*bloquear)(fd,F_UNLCK);
}
close (fd);
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
#include <stdio.h> /* e/s estandar */
#include <fcntl.h> /* atributos de fichero */
#include <sys/mode.h> /* valores de modos de fichero */
#include <sys/types.h> /* definiciones de tipos */
#include <unistd.h> /* constantes lseek */
main()
{
int fd;
int fd1;
int fd2;
int ret;
int retc;
short i;
char line[31];
/* abre archivo mifchero para lectura−escritura */
if ((fd=open("mifichero",O_RDWR,0644))!=0) {
/* crea archivo nuevofich con acceso de lectura−escritua */
```

```
if ((fd1=creat("nuevofich",0666))!=0) {
close(1);
/* escribe stdout en nuevofich */
fd2=fcntl(fd1,F_DUPFD,0);
printf("NOTA: esta lista es revision de mifichero \n");
/* salta 5 primeras entradas de michero */
ret=lseek(fd,(5*sizeof(line)),SEEK_SET);
/* lee 5 siguientes entradas de michero
y las escribe eb nuevofich */
for (i=0;i<5;i++) {
retc=read(fd,line,sizeof(line));
retc=write(fd1,line,sizeof(line));
}
close(fd1);
}
close(fd);
}
else
perror ("fichero mifichero inexistente");
}
```

```
#include <stdio.h> /* e/s estandar */
#include <sys/utsname.h> /* informacion del sistema */
#include <sys/types.h> /* definicion de tipos */
struct utsname *nptr;/* ptr. a la estr. de informacion del sistema */
main()
```

```
{

int ret;

nptr=(struct utsname *)malloc(sizeof(struct utsname));

/* asigna espacio para la estr. de informacion del sistema */

printf("mi id de usuario real es %d\n",getuid());

printf("mi id de usuario efectivo es %d\n",geteuid());

printf("mi id de grupo efectivo es %d\n",getegid());

ret=setuid(239);/* ojo a posibilidades de asignac. del euid Los codigos de retorno son 0 si es correcto, −1 si hay error */

printf("mi id de usuario es cambiado a %d\n",geteuid());

printf("mi id de grupo real es %d\n",getgid());

ret=setegid(211); /* ojo a posibilidades de asignac.del egid. Los codigos de retorno son 0 si es correcto, −1 si hay error */

printf("mi id de grupo efectivo es cambiado a %d\n",getegid());

ret=uname(nptr);

/* uname devuelve un numero no negativo si correcto, −1 si hay error. Se imprime solo el nombre del sistema de la estructura */

printf("El nombre del sistema actual es %s. \n", nptr−>sysname);

}

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Ejemplo de las llamadas al sistema link y unlink */

#include <stdio.h> /* cabecera de e/s estandar */

#include <sys/types.h> /* definicion de tipos */

#include <sys/stat.h> /* estado de fichero */

struct stat *stsptr;

main(int argc, char *argv[])

{

int retc;
```

```
stsptr=(struct stat *)malloc(sizeof(struct stat));
retc=chdir("/home/lda");
sync();
retc=stat(argv[1],stsptr);
printf("num. de enlaces para %s es: %d.\n", argv[1], stsptr−>st_nlink);
printf("enlazando con fichero %s...\n", argv[2]);
link(argv[1],argv[2]);
retc=stat(argv[2],stsptr);
printf("num. de enlaces ahora para %s es: %d.\n", argv[2], stsptr−>st_nlink);
printf("suprimiendo el enlace...\n");
unlink(argv[2]);
retc=stat(argv[1],stsptr);
printf("num. de enlaces vuelve a ser: %d.\n", stsptr−>st_nlink);
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Escribir el comando pwd */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/dir.h>
#define LARMAX 256
char ref[LARMAX];
struct stat buf1,buf2; /* para nodos−i*/
int d;
int encontrado;
struct direct catal; /* para contenido de los directorios*/
```

```
int n=sizeof(catal);/* tamaZo de el item catal*/
int indice=LARMAX;
main()
{ ref[LARMAX] ='\0';
for (;;)
{ if (stat(".",&buf1) <0)
{fprintf(stderr,"error en stat(.)\n");
exit(1);}
if ((d=open("..",0)) == −1 )
{fprintf (stderr,"error de apertura de..\n");
exit(2);}
if (stat("..",&buf2) <0 )
{fprintf(stderr,"error en stat(..)\n") ;
exit(3);}
if (chdir("..")<0)
{fprintf(stderr,"error sobre chdir(..)\n");
exit(4); }
if (buf1.st_dev == buf2.st_dev)
/* si los directorios . y .. estan en el mismo disco l\gico */
{ if(buf1.st_ino ==buf2.st_ino) escribir();
/*si est)n en el mismo disco, pero son diferentes */
encontrado=0;
while (!encontrado && read(d,(char *) &catal,n) == n)
if (catal.d_ino == buf1.st_ino) encontrado = 1;
if (!encontrado)
{fprintf(stderr,"error de lectura en ..\n");
```

```
exit(5);} }
else
{ encontrado =0;
while (!encontrado && read(d,(char *)&catal,n) == n)
{ stat(catal.d_name,&buf2);
if (buf2.st_dev == buf1.st_dev) encontrado=1; }
if (!encontrado)
{fprintf(stderr,"error de lectura en ..\n");
exit(5);} }
close(d);
tratamiento();
}
}
tratamiento()
{ static int n;
n = strlen(catal.d_name);
if ((indice −= n) < 0 )
{ fprintf(stderr,"referencia demasiado larga\n");
exit(6); }
strncpy(ref+(indice−−),catal.d_name,n);
strncpy(ref+indice,"/",1);
}
escribir()
/* escribe la cadena ref+indice*/
{
if(indice == LARMAX) printf("/");
```

```
else printf("%s\n",ref+indice);
exit(0);
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de la llamada al sistema statfs */
#include <stdio.h> /* cabecera de e/s estandar */
#include <sys/statfs.h> /* sistema de ficheros */
#include <sys/stat.h> /* estado del fichero */
struct statfs *sfsptr;
struct stat *stsptr;
main(int argc, char *argv[])
{
sfsptr=(struct statfs *)malloc(sizeof(struct statfs));
stsptr=(struct stat *)malloc(sizeof(struct stat));
if (argc==1) printf("ERROR: debe especificarse un fichero.\n"), exit(1);
if (stat(argv[1],stsptr) != 0)
printf("ERROR en llamada de stat.\n"), exit(1);
printf("num. de enlaces: %d\n",stsptr−>st_nlink);
printf("propietaro: %d\n",stsptr−>st_uid);
printf("ultimo acceso: %d\n",stsptr−>st_atime);
if (statfs(argv[1],sfsptr) != 0)
printf("ERROR en llamada de statfs.\n"), exit(1);
printf("nombre del sistema de ficheros: %s\n",sfsptr−>f_fname);
printf("total de bloques de datos: %d\n",sfsptr−>f_blocks);
printf("bloques libres: %d\n",sfsptr−>f_bfree);
}
```

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Llamada access: Chequeo de permisos de un fichero

par<metros: nombre del fichero y permisos a chequear */

#define ERROR −1

#include <stdio.h>

#include <errno.h>

main(argc,argv)

char argc;

char *argv[];

{ int permisos;

if (argc!=3)

{ printf ("((( Ndmero de par<metros incorrecto !!!\n");

return(1); }

permisos=atoi(argv[2]);

if (permisos>7)

{ printf("Valor %d demasiado grande : de 0 a 7\n");

return(2); }

if (access(argv[1],permisos)==ERROR)

{ if (errno=EACCES)

printf("Modo %d NO permitido sobre el fichero %s\n",

permisos,argv[1]);

else perror(argv[1]);

return(3); }

printf("Modo %d SI permitido sobre el fichero %s\n",

permisos,argv[1]);

return(0);
```

```
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Bloqueo de archivos */
#define LOCKDIR "/tmp/"
#define MAXTRIES 3
#define NAPTIME 5
int lock (name) /* conseguir cierre */
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 (0);
sleep (NAPTIME); }
if (fd==−1||close(fd)==−1) syserr("lock");
return(1);
}
void unlock(name) /* liberar */
char *name;
{ char *lockpath();
if (unlink(lockpath(name))==−1) syserr("unlock");
}
static char *lockpath(name)
/* generar un fichero para el bloqueo */
```

```
char *name;
{ static char path[20];
char *strcat();
strcpy(path,LOCKDIR);
return(strcat(path,name));
}
−−−−−−−−−−−−−−−−−
#define BUFSIZE 512
void copy2(from,to) /* copia de ficheros */
char *from,*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("read");
nwrite+=n;
} while(nwrite < nread); }
if (close(fromfd)==−1||close(tofd)==−1)
syserr("close");
}
−−−−−−−−−−−−−−−−−−−−−−−−−−
```

```
/* Conjunto de subrutinas que manejan los buffers autom<ticamente */

STREAM.H

#define SENOMEN 1001

#define SEINVAL 1002

#define BUFSIZE 512

typedef struct {

int fd; /*descriptor de archivo */

char dir; /* direccion: r o w */

int total; /* caracteres totales en buf */

int next; /* siguiente caracter en buf */

char buf[BUFSIZE];

} STREAM;

STREAM *Sopen();

int Sgetc();

BOOLEAN Sputc();

BOOLEAN Sclose();

#include "stream.h"

#include "stdio.h"

extern int errno;

STREAM *Sopen(path,dir) /*abrir corriente*/

char *path, *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); }
z−>fd=fd;
z−>total=z−>next=0;
return(z);
}
static BOOLEAN readbuf(z) /* llenar el buffer */
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(z) /* vaciar el buffer */
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 Sgets(z) /* leer car<cter */
STREAM * z;
{ int c;
if (z−>next >= z−>total && !readbuf(z))
return(−1);
return (z−>buf[z−>next++] & 0377);
}
BOOLEAN Sputc (z,c) /* escribir un car<cter */
STREAM *z;
char c;
{ z−>buf[z−>next++]=c;
if (z−>next>=sizeof(z−>buf)) return (writebuf(z));
return(TRUE);
}
BOOLEAN Sclose(z) /* cerrar la corriente */
STREAM *z;
{ int fd;
if (z−>dir=='w'&& !writebuf(z)) return(FALSE);
fd=z−>fd;
free(z);
```

```
return(close(fd) !=−1);
}
/* recodificaci\n de la rutina de copia de ficheros
utilizando este paquete */
void copy(from,to)
char *from,*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");
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Imprime un fichero hacia atr<s lRnea a lRnea */
void backward( path)
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("linea demasiado larga");
s[−−i]=c;
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");
}
```

−−−−−−−−−−−−−−−−−−−−−−−−−−

```
/* PequZa utilidad para el manejo de archivos */
#define ERROR −1
#define CIERTO 1
#define FALSO 0
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
main()
```

```
{ setbuf(stdout,NULL);
help();
mainloop();
}
static void mainloop() /* procesador de ordenes */
{ char path[50], cmd[10], shcmd[100];
while(1)
{ prompt("Orden: ",cmd);
if (strlen(cmd)>1) cmd[0]='\1'; /*fuerza el mensaje de comando
desconocido*/
switch(cmd[0])
{ case '\0':
case 'q': exit(0);
case 'a':
case 'm': chtime(path,cmd[0]); continue;
case 'f': prompt("Fichero: ",path);
if (access(path,0)==−1)
{ printf("%s No existe\n",path); continue; }
status(path); continue;
case 'o': chowner(path); continue;
case 'p': chperms(path); continue;
case 's': status(path); continue;
case '!': prompt("Orden SHELL: ", shcmd);
system (shcmd); continue;
case '?': help(); continue;
default: printf("Orden desconocida. ");
```

```
printf("Pulse >?< para pantalla de ayuda\n");
continue; } }
}
static void help() /*Pantalla de ayuda */
{ printf(" *** Ordenes de la utilidad UTOOLS ***\n");
printf(" ====================================\n");
printf(" a Cambiar fecha y hora de ultimo acceso\n");
printf(" f Seleccionar el fichero de trabajo\n");
printf(" m Cambiar fecha y hora de dltima modificaci\n\n");
printf(" o Cambiar propietario del fichero\n");
printf(" p Cambiar los permisos al fichero\n");
printf(" q Salir de la utilidad\n");
printf(" s Visualizar el estatus del fichero\n");
printf(" ! Ejecutar un comando UNIX\n");
printf(" ? Visualizar esta ayuda\n");
}
static void prompt (char *msg, char *entrada)
{ printf("\n%s",msg);
if(gets(entrada)==NULL) exit(0);
}
static void status(char *path) /* orden s */
{ struct stat sb;
if (stat(path,&sb)==−1)
{ perror("stat"); return; }
dspstatus(&sb);
}
```

```
static void dspstatus(struct stat *sbp) /* mostrar el estatus */
{ short isdevice=0;
struct passwd *pw, *getpwuid();
struct group *gr,*getgrgid();
char *nombre,*asctime();
struct tm *localtime();
if ((sbp−>st_mode & S_IFMT) == S_IFDIR)
printf("Directorio\n");
if ((sbp−>st_mode & S_IFMT) == S_IFBLK)
{ printf("Dispositivo Tipo Bloque\n");
isdevice=1; }
if ((sbp−>st_mode & S_IFMT) == S_IFCHR)
{ printf("Dispositivo Tipo Car<cter\n");
isdevice=1; }
if ((sbp−>st_mode & S_IFMT) == S_IFREG)
printf("Fichero Ordinario\n");
if ((sbp−>st_mode & S_IFMT) == S_IFIFO)
printf("Fichero FIFO\n");
if (isdevice) printf("Dispositivo numero : %d, %d \n",
(sbp−>st_rdev>>8)&0377,sbp−>st_rdev&0377);
printf("Reside en el dispositivo numero : %d, %d \n",
(sbp−>st_dev>>8)&0377,sbp−>st_dev&0377);
printf("Ndmero de I−nodo: %d, Links: %d, TamaZo: %ld\n",
sbp−>st_ino,sbp−>st_nlink,sbp−>st_size);
if ((pw=getpwuid(sbp−>st_uid))==NULL) nombre="????";
else nombre=pw−>pw_name;
```

```
printf("Propietario uid = %d; nombre = %s\n",sbp−>st_uid,nombre);
if ((gr=getgruid(sbp−>st_gid))==NULL) nombre="????";
else nombre=gr−>gr_name;
printf("Grupo gid = %d; nombre = %s\n",sbp−>st_gid,nombre);
if ((sbp−>st_mode & S_ISUID)==S_ISUID)
printf("Set−user−ID activado\n");
if ((sbp−>st_mode & S_ISGID)==S_ISGID)
printf("Set−group−ID activado\n");
if ((sbp−>st_mode & S_ISVTX)==S_ISVTX)
printf("Salva del segmento de texto activadao\n");
printf("Permisos: %o\n",sbp−>st_mode & 0777);
printf("Ultimo acceso .............: %s",
asctime (localtime(&sbp−>st_atime)));
printf("Ultima modificaci\n .......: %s",
asctime (localtime(&sbp−>st_mtime)));
printf("Ultimo cambio de estatus ..: %s",
asctime (localtime(&sbp−>st_ctime)));
}
static void chtime (path, orden) /* \rdenes "a" y "m" */
char *path, *orden;
{ char atime[20];
long seg,timecvt();
struct stat sb;
struct utimbuf {
time_t actime;
time_t modtime; } tb;
```

```
if (stat(path,&sb)==ERROR)
{ perror("stat");
return; }
prompt ("Entre el valor del tiempo (YYMMDDhhmmss) :", atime=);
if ((seg=timecvt(atime)) <=0) return;
switch(orden)
{ case 'a': tb.actime=seg;
tb.modtime=sb.st_mtime;
break;
case 'm': tb.actime=sb.st_atime;
tb.modtime=seg; }
if (utime(path,&tb)==ERROR) perror ("utime");
}
static void chowner (path) /* orden "o" */
char *path;
{ char oname[20],gname[20];
int owner,group;
struct passwd *pw, *getpwnam();
struct group *gr, *getgrnam();
prompt ("Nombre del propietario :",oname);
if ((pw=getpwnam(oname))==NULL)
{ printf("Nombre de propietario desconocido\n");
return; }
owner=pw−>pw_uid;
prompt ("Nombre del grupo :",gname);
if ((gr=getgrnam(gname))==NULL)
```

```
{ printf("Nombre de grupo desconocido\n");
return; }
group=gr−>gr_gid;
if (chown(path,owner,group)==ERROR) perror("chown");
return;
}
static void chperms (path) /* orden "p" */
char *path;
{ char smode[20];
int mode;
prompt ("Protecciones (en 4 digitos octales)", smode);
if (sscanf(smode,"%o",&mode)!=1)
{ printf("Modo invalido\n");
return; }
if (chmod(path,mode)==ERROR) perror("chmod");
return;
}
static long timecvt (atime)
char *atime;
/* Conversi\n de YYMMDDhhmmss a formato interno */
{ long tm;
int i,n,dias,zona;
char s[3], *getenv(),*tz;
int esbisiesto;
extern struct tm *localtime();
if (strlen(atime)!=12)
```

```
{ printf ("Fecha y hora en 12 digitos : YYMMDDhhmmss\n");

return (0); }

for (i=0; i<12; i++)

if (atime[i] <'0'|| atime[i] >'9')

{ printf"La fecha/hora tiene caracteres no numJricos\n");

return (0); }

if ((tz=getenv("TZ"))==NULL) zona=5;

else zona=atoi(&tz[3]);

s[2]='\0';

/* p<g 51 de Advanced UNIX Programming */

return;

}
```

EJEMPLOS DE UTILIZACIÓN DE LAS LLAMADAS AL SISTEMA DEL SUBSISTEMA DE PROCESOS EN UNIX¡Error! Marcador no definido.

```
/* PequeZa shell */

#include <stdio.h>

#include <string.h>

#define MAXARG 20

#define CIERTO 1

#define FALSO 0

#define PS1 "Orden :"

static void ejecuta (int argc, char *argv[]);

static int getargs (int *argcp, char *argv[], int max);

void main(void)

{ int argc;

char *argv [MAXARG+1];
```

```
while (CIERTO) {
printf("%s",PS1);
if (!getargs(&argc,argv,MAXARG) || argc==0)
continue; /* Volver a leer una orden */
ejecuta (argc,argv); }
printf("FIN DE EJECUCION \N"); }
static int getargs (int *argcp, char *argv[], int max)
{ static char cmd [100];
char *cmdp;
int i;
if (gets(cmd)==NULL) return (FALSO);
cmdp=cmd;
for (i=0;i<=max;i++)
{ if((argv[1]=strtok(cmdp, " \t"))==NULL) break;
cmdp=NULL; }
if (i>max) {
printf("Demasiados argumentos \n");
return(FALSO); }
*argcp=i;
return(CIERTO); }
static void ejecuta (int argc, char *argv[])
{ execvp (argv[0], argc);
printf("No puedo ejecutarlo \n");
}
```

```
/* Ejemplo de fork() */
```

```
void main(void)
{
int pid;
printf("Empiezo un proceso \n");
pid=fork();
if(pid>0) printf("Esto es el proceso PADRE : pid = %d \n",pid);
else printf("Esto es el proceso HIJO : pid = %d \n",pid);
return;
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de fork() */
#include <stdio.h>
#define ERROR −1
main(void)
{ int pid;
switch(fork()) {
case ERROR: errorsistema ("fork");
return (1);
case 0 : system ("ps −f");
printf("Fin del proceso HIJO \N");
exit(1);
default : if (wait(NULL)==ERROR)
errorsistema("wait");
printf("Fin del proceso PADRE\n"); }
return 0;
}
```

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Ejemplo de una pequeZa shell utilizando fork() */

#include <stdio.h>

#include <string.h>

#define ERROR −1

#define MAXARG 20

#define CIERTO 1

#define FALSO 0

#define PS1 "Orden :"

static void ejecuta (int argc, char *argv[]);

static int getargs (int *argcp, char *argv[], int max);

void main(void)

{

int argc;

char *argv [MAXARG+1];

while (CIERTO) {

printf("%s",PS1);

if (!getargs(&argc,argv,MAXARG) || argc==0)

continue; /* Volver a leer una orden */

ejecuta (argc,argv); }

printf("FIN DE EJECUCION \N");

}

static int getargs (int *argcp, char *argv[], int max)

{

static char cmd [100];

char *cmdp;
```

```
int i;
if (gets(cmd)==NULL) return (FALSO);
cmdp=cmd;
for (i=0;i<=max;i++)
{ if((argv[1]=strtok(cmdp, " \t"))==NULL) break;
cmdp=NULL; }
if (i>max) {
printf("Demasiados argumentos \n");
return(FALSO); }
*argcp=i;
return(CIERTO);
}
static void ejecuta (int argc, char *argv[])
{
switch(fork()) {
case ERROR: errorsistema ("fork");
return (1);
case 0 : execvp (argv[0], argc);
printf("No puedo ejecutarlo \n");
exit(1);
default : if (wait(NULL)==ERROR)
errorsistema("wait"); }
return 0;
}
```

```
/* Ejemplo de exec() */
```

```
#include <stdio.h>

main()

{

execl ("/bin/echo", "echo", "Hola amigo\n", "esto", "es una lata",NULL);

return (0);

}
```

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* El siguiente programa cuenta el ndmero de ficheros no directorios, creando un nuevo proceso por cada fichero que encuentra en los directorios que va leyendo. Cada proceso hijo verifica el tipo de fichero, si es un directorio comienza a generar hijos por cada entrada. Si el fichero no es directorio termina devolviendo uno a su proceso padre. */

```
#include <stdio.h>

#include <sys/file.h>

#include <sys/dir.h>

#include <sys/stat.h>

main(int argc, char *argv[])

{

long count;

count=processFIle(argv[1]);

printf("Ndmero total de ficheros no directorios %ld\n",count);

return(0);

}

long processfile(char *name)

{

struct stat statbuf;

mode_t mode;

if(stat(name,&statbuf)==−1)return(0);

mode=statbuf.st_mode;
```

```
if (S_ISDIR(mode)) return(processDirectory(name));
else return(1);
}
long processDirectory(char *dirname)
{
int fd,children,i,charsRead,chilspid,status;
long count,totalcount;
char filename[100];
struct direct dirEntry;
fd=open(dirname,O_RDONLY);
children=0;
while(1)
{ charsRead=getdents (fd,&dirEntry,sizeof(struct direct));
if(!charsRead) break;
if(strcmp(dirEntry.d_name,".")&&strcmp(dirEntry.d_name,
"..");
{ if(!fork())
{ sprintf(filename,"%s/%s",dirname,dirEntry.d_name);
count=processfile(filename);
exit(count); }
else ++children; }
lseek(fd,dirEntry.d_off,SEEK_SET); }
close (fd);
totalCount=0;
for(i=0;i<=children;i++)
{ childPid=wait(&status);
```

```
totalCount+=(status>>8); }
return(totalCount);
}
--------------------------------------
/* Control del tiempo de ejecuci\n de un proceso hijo por parte del padre */
main(int argc, char * argv[])
{
int pid;
signal(SIGCHLD, childhandler);
pid=fork();
if (pid==0)
{ execvp(argv[2],&argv[2]);
perror("limit"); }
else
{ sscanf(argv[1],"%d",&delay);
sllep(delay);
printf("El hijo %d excedi\ su lRmite y va a ser matado\n",pid);
kill(pid,SIGINT); }
}
childhandler (void)
{ int childpid, childstatus;
childpid=wait(&childstatus);
printf("El hijo %d termin\ en %d segundos\n",childpid,
delay);
exit(0);
}
```

```
----------

/* Uso de las seZales SIGSTOP y SIGCONT para suspender y continuar un proceso. */

main()

{

int pid1,pid2;

pid1=fork();

if (pid1==0)

{ while(1)

{ printf("%d est< vivo\n",pid1);

sleep(1); } }

pid2=fork();

if (pid2==0)

{ while(1)

{ printf("%d est< vivo\n",pid2);

sleep(1); } }

sleep(3);

kill(pid1,SIGSTOP);

sleep(3);

kill(pid1,SIGCONT);

sleep(3);

kill(pid1,SIGINT);

kill(pid2,SIGINT);

}

----------

/* at.c simulaci\n de la orden at de UNIX

versi\n para 4.3BSD
```

```
Recibe \rdenes para ejecutarlas en background a una hora
determinada. Uso: $at hh:mm:ss linea_de_\rdenes */
#include <stdio.h>
#include <time.h>
#include <signal.h>
#define SEGDIA 86400
#define hhmmss_ss (_hh,_mm,_ss) ((long)(_hh)*3600+ \
(_mm)*60+(_ss))
#define ss_hhmmss (_seg,_hh,_mm,_ss) \
{ _hh=(int)((_seg)/3600); \
_mm=(int)(((_seg)−(long)_hh*3600)/60); \
_ss=(int)((_seg)−(long)_hh*3600−_mm*60); }
void sigalarmhandler() {}
main(argc,argv)
int argc;
char **argv;
{ int hh,mm,ss,pid;
if (argc<3)
{ fprintf(stderr,"Uso: %s hh:mm:ss linea_de_\rdenes\n",argv[0]);
exit(−1); }
if (sscanf(argv[1],"%d:%d:%d",&hh,&mm,&ss)!=3)
{ fprintf(stderr,"Formato de hora err\neo: %s\n",argv[1]);
exit(−1); }
else if (hh<0||hh>23||mm<0||mm>59||ss<0||ss>59)
{ fprintf(stderr,"Hora fuera de rango: %s\n",argv[1]);
exit(−1); }
```

```
if ((pid=fork())==−1)
{ perror(argv[0]);
exit(−1); }
else if (pid=0)
{ long totalseg;
time_t t;
struct tm *tm;
struct itimerval temporizador, temporizador_ant;
temporizador.it_value.tv_usec=0;
temporizador.it_interval.tv_sec=0;
temporizador.it_interval.tv_usec=0;
signal (SIGALRM,sigalarmhandler);
time(&t);
tm=localtime(&t);
totalseg=hhmmss_ss(hh,mm,ss)−hhmmss_ss(tm−>tm_hour,
tm−>tm_min,tm_>tm_sec);
if (totalseg<0) totalseg+=SEGDIA;
temporizador.it_value.tv_sec=totalseg;
setitimer(ITIMER_REAL,&temporizador,&temporizador_ant);
pause(); /* ejecutar cuando expire el temporizador */
execvp(argv[2],&argv[2]);
}
else exit(0); /*padre*/
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Uso de alarm() */
```

```
#include <stdio.h>

#include <signal.h>

main(int argc, char *argv[])

{

int valor, tiempo, fact1,fact2,accion();

tiempo=atoi(argv[1]);

fact1=atoi(argv[2]);

fact2=atoi(argv[3]);

printf("Tiene Ud. %d segundos para resolver \n", tiempo);

printf("el producto de : %d por %d \n",fact1,fact2);

alarm(tiempo);

signal(SIGALRM,accion);

printf("Su solucion : ");

scanf("%d",&valor);

printf("Ha tardado Ud. %d segundos n",tiempo−alarm(0));

if (valor==fact1*fact2) printf("La respuesta es correcta !!!");

else printf("La respuesta es incorrecta !!!");

return (0);

}

int accion()

{ printf("\nSu tiempo se ha agotado. Lo siento !!!\n");

exit(0);

}

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/*

DEFINICION ... Perfil de un proceso: a la hora de realizar una aplicacion nos puede interesar hacer estadisticas para conocer los tiempos de ejecucion de las diferentes funciones con que cuenta la aplicacion. La
```

forma de realizar dicha estadistica, es creando un perfil.En UNIX los perfiles se crean mediante la llamada 'profil', cuya declaracion es la siguiente:

void profil(buf,bufsiz,offset,scale)

char *buf;

int bufsiz,offset,scale;

buf: apunta a un area de memoria cuya longitud viene dada por bufsiz

Este programa muestra como crear un perfil de ejecucion de un proceso. Tiene tres funciones: f,g,fin. Mediante la llamada profil indicamos al sistema que haga un profiling de las tres funciones que nos interesen. */

#include <stdio.h>

#include <signal.h>

#define MAXF 1000

#define MAXG 5000

int main(),f(),g();

#define FINICIO ((int)main) /* inicio de la zona de c\digo a perfilar */

#define FFIN ((int)fin) /* final zona a perfilar */

/* Macros para acceder a los offset inicial y final de la funcion

que se controla con el elemento x del array funciones */

#define inicio_funcion(x) ((funciones[(x)].inicio−FINICIO)/sizeof(int))

#define fin_funcion(x) ((funciones[(x)].fin−FINICIO)/sizeof(int))

struct funcion

{

char *nombre;

int inicio,fin;

unsigned contador;

};

struct funciones[]

{

```
"main",(int)main,(int)f−1,0,
"f",(int)f,(int)g−1,0,
"g",(int)g,(int)fin−1,0
};
#define total_funciones (sizeof(funciones)/sizeof(struct funcion))
main (int argc,char* argv[])
{
bufsiz=FFIN−FINICIO;
if ((buf=(int *)calloc(bufsiz,sizeof(int)))==null)
{
perror(argv[0]);
exit(−1);
}
profil (buf,bufsiz,FINICIO,0xffff);
signal(SIGALRM,fin);
alarm(10);
for(;;)
{
f();
g();
}
}
/* Funciones que simulan un tratamiento genJrico de informaci\n */
int f()
{
long i=Irand48()%MAXF;
```

```
while(i−−);
}
int g()
{
long i=Irand48()%MAXG;
while(i−−);
}
/* Esta funci\n analiza el buffer de perfiles para actualizar los
contadores que hay en el array de funciones. Se activa cada 10 seg */
fin()
{
int i,j;
double total;
static int cnt=0;
for (i=0;i<bufsiz/sizeof(int);i++)
if (buf[i])
for (j=0;j<total_funciones;j++)
if (i>=inicio_funcion(j) && i<=fin_funcion(j))
funciones[j].contador +=buf[i];
total=0;
for(i=0;i<total_funciones;i++)
total +=funciones[i].contador;
for (i=0;i<total_funciones;i++)
printf("%s=%d=%g%%\n",funciones[i].nombre,funciones[i].contador,
funciones[i].contador/total*100);
printf("\n");
```

```
if(++cnt<5)

signal(SIGALRM,fin);

else

exit(0);

alarm(10);

}
```

−−

/* CONTABILIDAD ... : cuando queremos llevar la contabilidad de los usuarios que utilizan el sistema (que ordenes utilizan, que tiempos emplean, etc. UNIX dispone de ordenes para habilitar o no la contabilidad, estas son 'accton' (habilita/deshabilita la contabilidad) y 'acctcom' (visualizar el contenido de un fichero de contabilidad)

```
int acct(path)

char *path;
```

path: puntero al path name del fichero sobre el que se van a escribir los registros de contabilidad.

Si path es NULL entonces la contabilidad se habilita, en caso contrario se deshabilita.

Este programa lleva la contabilidad del sistema. Asocia bajo un mismo programa las \rdenes standard accton y acctcom.

FORMA DE USO ... : $capit82 [opciones] [fichero]

−e: habilita la contabilidad

−d: deshabilita la contabilidad

−r: muestra por pantalla el contanido del fichero de contabilidad

−f fichero: en el caso de que estJn presentes −e o −r, sirve para especificar un fichero diferente al de por defecto.

*/

```
#include <stdio.h>

#include <sys/types.h>

#include <sys/acct.h>

#include <pwd.h>

#include <time.h>
```

```
#include <sys/param.h>
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
#define PERMISOS 0644
char opciones[]="derf:";
char error="[−der] [−f fichero]";
char *nombre_programa;
/* Analiza los par<metros de la lRnea de \rdenes */
main(int argc,char *argv[])
{
int c;
extern char *optarg;
extern int optind;
int errfig=0;
char *home,acct_path[256];
enum{NINGUNA,ACTIVAR,DESACTIVAR,LEER} accion=NINGUNA;
FILE *f;
struct acct act;
struct passwd *pw,*getpwuid();
struct tm *tm;
time_t tfin;
char *getenv(),*nombre_tty();
nombre_programa=argv[0];
while ((c=getopt(argc,argv,opciones)) !=EOF)
switch(c)
```

```
{
case 'd':
accion=DESACTIVAR;
break;
case 'e';
accion=ACTIVAR;
home=getenv("HOME");
sprintf(acct_path,"%s/adm/pacct",home);
break;
case 'r':
accion=LEER;
home=getenv("HOME");
sprintf(acct_path,"%s/adm/pacct",home);
break;
case 'f':
sprintf(acct_path,"%s",optarg);
break;
case '?':
errflg++;
}
if (errflg)
{
fprintf(stderr,"forma de uso:%s%s\n",argv[0],error);
exit(−1);
}
switch(accion)
```

```
{
case DESACTIVAR:
if (acct(NULL)==−1)
{
perror(argv[0]);
exit(−1);
}
else
{
printf("contabilidad desactivada");
exit(0);
}
case ACTIVAR:
close(creat(acct_path,PERMISOS));
if(acct(acct_path)==−1)
{
perror(acct_path);
exit(−1);
}
else
{
printf("contabilidad activada sobre %s\n",acct_path);
exit(0);
}
case LEER:
if ((f=fopen(acct_path,"r"))==NULL)
```

```
{
perror(acct_path);
exit(−1);
}
printf("%−9s%−8s%−8s%−8s%−8s%−5s%−8s\t%−5s\t%−5s\n","","","",
"HORA","HORA","CARACT","T.REAL","T.CPU");
printf("%−9s%−8s%−8s%−8s%−8s%−5s%−8s\t%−5s0t%−5s\n","ORDEN",
"USUARIO","TTY","INICIO","FIN","L/E","L/E","(s.)","(s.)");
while (fread(&act,sizeof(act),1,f)==1)
{
printf("%−c",(act.ac_flag&ASU)?'#':");
printf("%−8s",act.ac_comm);
pw=getpwuid(act.ac_uid);
printf("%−8s",pw−>pw_pw_name);
printf("−8s",nombre_tty(act.ac_tty));
tm=localtime(&act.ac_btime);
printf("%02d:%02d:%02d",tm−>tm_hour,tm−>tm_min,tm−>tm_sec);
tfin=act.ac_btime+act.ac_etime/HZ;
tm=localtime(&tfin);
printf("%02d:%02d:%02d",tm−>tm_hour,tm−>tm_min,tm−>tm_sec);
printf("%5d",act.ac_rw);
printf("%8d\t",act.ac_io);
printf("%2.2f\t",(float)act.ac_etime/HZ);
printf("%2.2f\n",(float)(act.ac_utime+act.ac_stime)/HZ);
}
fclose(f);
```

```
exit(0);

case NINGUNA:

fprintf(stderr,"Forma de uso: %s%s\n",argv[0],error);

exit(−1);

}

}

/* A partir de un ndmero de dispositivo, devuelve un puntero a una cadena de caracteres, que contiene el nombre del fichero de dispositivo al que pertenece el ndmero. */

char *nombre_tty(dev)

dev_t dev;

{

DIR *dir;

struct dirent *dirent;

char tty_path[256];

static char n_tty[256];

struct stat buf;

if ((dir=opendir("/dev"))==NULL)

{

perror(nombre_programa);

exit(−1);

}

seek(dir,2);

while ((dirent=readdir(dir))!=NULL)

{

sprintf(tty_path,"/dev/%s",dirent−>d_name);

if(stat(tty_path,&buf)==−1)

{
```

```
perror(tty_path);

exit(−1);

}

if(buf.st_rdev==dev)

{

sprintf(n_tty,"%s",dirent−>d_name);

break;

}

}

closedir(dir);

return(n_tty);

}
```

−−−−−−−−−−−−−−−−−−−−−−−−−

/* DEPURACION DE PROGRAMAS

Un depurador es un programa que permite controlar la ejecuci\n de otro programa. UNIX suministra dos depuradores standard: adb y sdb.

abd: depurador de bajo nivel que permite depurar programas

escritos en ensamblador.

sdb: para programas escritos en C.

Ambos estan inplementados en base a la llamada 'ptrace'(llamada al sistema que permite la comunicaci\n entre dos procesos: el padre, que va a actuar como depurador, y el hijo, que va a actuar como proceso depurado)

Este programa implementa el ndcleo de un depurador de bajo nivel. Las funciones que lleva a cabo son elementales:

−Ejecutar paso a paso

−Examinar una zona de memoria

−Modificar una zona de memoria

La forma de uso es la siguiente:

$ capit83 [nom_programa] */

```
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#define PT_SETTRC 0
#define PT_RDUSER 2
#define PT_WDUSER 5
#define PT_CONTIN 7
#define PT_EXIT 8
#define PT_SINGLE 9
char *ayuda[] =
{
"\n",
"OPCIONES POSIBLES\n",
"p direccion\t muestra el contenido de direccion \n",
"s direccion valor\t escribe valor de direccion\n",
"t[#]\t\t ejecuta # instrucciones maquina\n",
"c\t\t continua la ejecucion del programa a depurar\n",
"!orden\t\t ejecuta una orden del sistema operativo\n",
"q\t\t salir del depurador\n",
"?\t\t muestra el menu de ayuda\n",
"\n"
};
#define MAX_AYUDA (sizeof(ayuda)/sizeof(char *))
#define EQ(str1,str2) (strcmp((str1)(str2))==0)
main(int argc,char *argv[])
{
```

```
int pid;
char prog[256];
if(argc<2)
strcpy(prog,"a.out");
else
strcpy(prog,argv[1]);
if((pid=fork())==0)
{
ptrace(PT_SETTRC);
/* cargamos el codigo del programa a depurar */
execcvp(prog,&argv[1],0);
perror(prog);
exit(127);
}
/* proceso padre, se encarga de depurar */
printf("proceso hijo: %d\n",pid);
for(;;)
{
char str_ant[256],orden;
int total,valor,direccion,i,estado;
enum {NO,SI} repetir;
/* esperamos a que el proceso hijo reciba la seZal SIGTRAP */
wait(NULL);
/* el proceso padre lee de la entrada standard una orden */
leer_orden:
do
```

```
{
printf("PID(%d)?==>ayuda<",getpid());
fflush(stdout);
gets(str_act);
if(EQ(str_act,""))
{
strcpy(str_act,str_ant);
repetir=SI;
}
else
{
strcpy(str_ant,str_act);
total=sscanf(str_act,"%c%x%x%x",&orden,&direccion,&valor);
repetir=NO;
}
}while(total<=0);
/* analisis de la orden introducida */
switch(orden)
case 'p':
if (repetir)
++direccion;
else if(total!=2)
{
printf("[%s]incorrecto\n",str_act);
goto leer_orden;
}
```

```
if ((valor=ptrace(PT_RDUSER,pid,direccion))==−1)
printf("direccion 0x%xincorrecta\n",direccion);
else
printf("(0x%x)=0x%x=%d\n",direccion,valor,valor);
break;
case 's':
if (total!=3)
{
printf("[%s]incorrecto\n",str_act);
goto leer_orden;
}
if (ptrace(PT_WDUSER,pid,direccion,valor)!=valor)
printf("direccion 0x%x incorrecta\n",direccion);
else
printf("(0x%x)=0x%x=%d\n",direccion,valor,valor);
break;
case 't':
if (total==1)
valor=1;
else
valor=direccion;
for (i=0;i<valor;i++)
if(ptrace(PT_SINGLE,pid,1,0)==−1)
perror(argv[0]);
break;
case 'c':
```

```
if (ptrace(PT_CONTIN,pid,1,0)==−1)
perror(argv[0]);
break;
case '!':
system(str_act+1);
goto leer_orden;
case 'q':
if (kill(pid,0)==−1)
exit(0);
if (ptrace(PT_EXIT,pid)==−1)
perror(argv[0]);
exit(0);
case '?':
for(valor=0;valor<MAX_AYUDA;valor++)
printf(ayuda[valor]);
goto leer_orden;
default:
printf("%s incorrecto\n",str_act);
goto leer_orden;
}
}
}
```

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

```
/* Con este programa vamos a mostrar el uso del depurador.
Los puntos de parada son aquellos en los que el proceso recibe
la seZal SIGTRAP. */
```

```
#include <signal.h>

int i;

main()

{

int pid=getpid();

printf("pid(%d)&i=0x%x\n",pid,(int)&i);

printf("pid(%d)i=0x%x\n",pid,i);

i=100;

kill(getpid(),SIGTRAP);/* punto de parada */

printf("pid(%d)i=0x%x\n",pid,i);

kill(getpid(),SIGTRAP); /* punto de parada */

}
```

```
/* TUBERIAS SIN NOMBRE

Este programa nos ayuda a ilustrar el envio de mensajes entre un proceso emisor y un proceso receptor a
travJs de una tuberia sin nombre */

#include <stdio.h>

#define MAX 256

main()

{

int tuberia[2];

int pid;

char mensaje[MAX];

/* Creacion de la tuberia sin nombre */

if (pipe(tuberia)==-1)

{

perror("pipe");
```

```
exit(−1);

}

/* creacion del proceso hijo */

if((pid=fork())==−1)

{

perror("fork");

exit(−1);

}

else if(pid==0);

{

/* El proceso receptor (hijo) se encarga de leer un mensaje

de la tuberia y presentarlo en pantalla */

while (read(tuberia[0],mensaje,MAX)>0 && strcmp(mensaje,"FIN") !=0)

printf("PROCESO RECEPTOR.MENSAJE:%s\n",mensaje);

close(tuberia[0]);

close(tuberia[1]);

exit(0);

}

else

{

/* El proceso emisor (padre) se encarga de leer un mensaje de la entrada standard y escribirlo en la tuberia
para que el proceso hijo lo reciba */

while(printf("PROCESO EMISOR.MENSAJE:")!=0 && gets(mensaje) !=NULL &&
write(tuberia[1],mensaje,strlen(mensaje)+1)>0

&& strcmp(mensaje,"FIN")!=0)

close (tuberia[0]);

close (tuberia[1]);
```

```
exit(0);

}

}
```

/* COMUNICACION BIDIRECCIONAL CON TUBERIAS SIN NOMBRE

Programa para ilustrar el envio de mensajes entre un proceso emisor y otro receptor a traves de dos tuberias sin nombre. El proceso emisor va a pedir un mensaje que le va a enviar al proceso receptor. Cuando el proceso receptor haya presentado el mensaje por pantalla, va a enviar al proceso emisor otro mensaje para indicarle que esta listo y que puede pedir otro mensaje al usuario */

```
#include <stdio.h>

#define MAX 256

main()

{

int tuberia1[2],tuberia2[2];

int pid;

char mensaje[MAX];

/* Creacion de las tuberias de comunicacion */

if (pipe(tuberia1)==−1 || pipe(tuberia2)==−1)

{

perror("pipe");

exit(−1);

}

/* creacion del proceso hijo */

if((pid=fork())==−1)

{

perror("fork");

exit(−1);

}
```

```
else if (pid==0)
{
/* El proceso hijo (receptor) se va a encargar de leer un
mensaje de la tuberia y presentarlo por pantalla. Al
recibir el mensaje FIN, el proceso termina */
while(read(tuberia1[0],mensaje,MAX)>0 && strcmp(mensaje,"FIN")!=0)
{
printf("PROCESO RECEPTOR.MENSAJE:%s\n",mensaje);
/* Le decimos al emisor que estamos listos para
recibir otro mensaje */
strcpy(mensaje,"LISTO");
write(tuberia2[1],mensaje,strlen(mensaje)+1);
}
close(tuberia1[0]);
close(tuberia1[1]);
close(tuberia2[0]);
close(tuberia2[1]);
exit(0);
}
else
{
/* El proceso emisor (padre) se encarga de leer un mensaje
de la entrada standard y escribirlo en la tuberia, para que
el proceso hijo lo reciba. Al recibir FIN los dos procesos
acaban */
while(printf("PROCESO EMISOR.MENSAJE:")!=0 && gets(mensaje) !=NULL &&
write(tuberia1[1],mensaje,strlen(mensaje)+1)>0
```

```
&& strcmp(mensaje,"FIN")!=0)
do
{
read(tuberia2[0],mensaje,MAX);
}while(strcmp(mensaje,"LISTO")!=0);
close(tuberia1[0]);
close(tuberia1[1]);
close(tuberia2[0]);
close(tuberia2[1]);
exit(0);
}
}
```

```
/* Este programa es un ejemplo para mostrar el uso que el shell
hace de las tuberias.
La forma de uso es la siguiente:
$ capit94 prog1 prog2
que es equivalente a:
$ prog1 | prog2 */
#include <stdio.h>
main(int argc,char *argv[])
{
int tuberia[2];
int pid;
if (argc<3)
{
```

```
fprintf(stderr,"Forma de uso:%s prog1 prog2\n",argv[0]);
exit(−1);
}
if (pipe(tuberia)==−1)
{
perror(argv[0]);
exit(−1);
}
if ((pid=fork())==−1)
{
perror(argv[0]);
exit(−1);
}
else if (pid==0)
{
close(0);
dup (tuberia[0]);
close(tuberia[0]);
close(tuberia[1]);
standard de entrada */
execlp(argv[2],argv[2],0);
}
else
{
close(1);
dup(tuberia[1]);
```

```
close(tuberia[0]);

close(tuberia[1]);

/* ejecucion de prog1 */

execlp(argv[1],argv[1],0);

}

exit(0);

}
```

```
/* TUBERIAS CON NOMBRE (FIFO)

Este programa simula conversaciones por radio. Programa para llamar.

La forma de uso es:

$ capit94 usuario

El programa exige que la variable de entorno LOGNAME este correctamente inicializada con el nombre de
usuario */

#include <stdio.h>

#include <string.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <utmp.h>

#define MAX 256

#define EQ(str1,str2) (strcmp(str1),(str2))==0)

int fifo12,fifo21;

char nom_fifo12[MAX],nom_fifo21[MAX];

char mensaje[256];

main(int argc,char *argv[])
```

```
{
int tty;
char terminal[MAX],*logname,*getenv();
struct utmp *utmp,getutent();
void fintransmision();
if (argc !=2)
{
fprintf(stderr,"forma de uso: %s user\n",argv[0]);
exit(−1);
}
while ((utmp=getutent())!=NULL && strncmp(utmp−>ut_user,argv[1],8)!=0);
if (utmp==NULL)
{
printf("EL USUARIO %s NO ESTA EN SESION\n",argv[1]);
exit(0);
}
sprintf(terminal,"/dev/%s",utmp−>ut_line);
if ((tty=open(terminal,O_WRONLY))==−1)
{
perror(terminal);
exit(−1);
}
logname=getenv("LOGNAME");
sprintf(mensaje,"\n\tLLAMADA PROCEDENTE DEL USER %s\07\07\07\nRESPONDER
ESCRIBIENDO:responder_a $s\n\n",logname,logname);
write(tty,mensaje,strlen(mensaje)+1);
close(tty);
```

```
signal(SIGINT,fintransmision);
sprintf(nom_fifo12,"/usr/tmp/%s_%s",logname,argv[1]);
sprintf(nom_fifo21,"/usr/tmp/%s_%s",argv[1],logname);
unlink(nom_fifo12);
unlink(nom_fifo21);
if (mknod(nom_fifo12,S_IFIFO | 0666,0)==−1 || mknod(nom_fifo21,S_IFIFO | 0666,0)==−1)
{
perror(argv[0]);
exit(−1);
}
if ((fifo12=open(nom_fifo12,O_WRONLY))== −1 ||
(fifo21=open(nom_fifo21,O_RDONLY))==−1)
{
perror(argv[0]);
exit(−1);
}
printf("LLAMADA ATENDIDA\07\07\07\n");
do
{
printf("<==");
gets(mensaje);
write(fifo12,mensaje,strlen(mensaje)+1);
if (EQ(mensaje,"cambio"))
do
{
read(fifo21,mensaje,MAX);
```

```
printf("==> %s\n",mensaje);
}while(!EQ(mensaje,"cambio"));
}while(!EQ(mensaje,"cambio"));
/*FIN DE LA TRANSMISION*/
printf(FIN DE TRANSMISION");
close(fifo12);
close(fifo21);
exit(0);
}
void fintransmision(int sig)
{
sprintf(mensaje,"corto");
write(fifo12,mensaje,strlen(mensaje)+1);
printf("fin de transmision");
close(fifo12);
close(fifo21);
exit(0);
}
```

```
/* Programa de simulacion de conversaciones por radio. Programa para responder.
Forma de uso:
$ capit95 usuario
Este programa exige que la variable de entorno LOGNAME este bien inicializada con el nombre de usuario */
#include <stdio.h>
#include <signal.h>
#include <fcntl.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#define MAX 256
#define EQ(str1,str2) (strcmp((str1),(str2))==0)
int fifo12,fifo21;
char nom_fifo12[MAX],nom_fifo21[MAX],mensaje[MAX];
main(int argc,char *argv[])
{
char *logname,*detenv();
void fintransmision();
if (argc!=2)
{
fprintf(stderr,"forma de uso:%s user\n",argv[0]);
exit(−1);
}
logname=getenv("LOGNAME");
signal(SIGINT,fintransmision);
sprintf(nom_fifo12,"/usr/tmp/%s_%s",argv[1],logname);
sprintf(nom_fifo21,"/usr/tmp/%s_%s",logname,argv[1]);
if ((fifo12=open(nom_fifo12,O_WRONLY))==−1) ||
(fifo21=open(nom_fifo21,O_WRONLY))==−1)
{
perror(nom_fifo21);
exit(−1);
}
do
```

```
{
printf("==>");fflush(stdout);
read(fifo12,mensaje,MAX);
printf("%s\n",mensaje);
if (EQ(mensaje,"cambio"))
do
{
printf("<==");
gets(mensaje);
write(fifo21,mensaje,strlen(mensaje)+1);
}while ()!EQ(mensaje,"cambio"));
}while (!EQ(mensaje,("corto"));
printf("FIN TRANSMISION");
close(fifo12);
close(fifo21);
exit(0);
}
void fintransmision(sig)
{
sprintf(mensaje,"corto");
write(fifo21,mensaje,strlen(mensaje)+1);
printf("FIN TRANSMISION");
close(fifo12);
close(fifo21);
exit(0);
}
```

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Empleo del polling o interrogaci\n peri\dica

Este programa emula comunicaciones telefonicas. Vamos a manejar dos fuentes de datos, el teclado y una tuberia de comunicaciones. Como la comunicacion es bidireccional, necesitamos dos tuberias. Basicamente el programa lee datos del teclado y escribe en una de las tuberias, o lee datos de la segunda tuberia y los escribe en la pantalla. La periodicidad con la que preguntaremos la existencia de datos de entrada sera de 1 segundo. Primero preguntaremos por el teclado y luego por la tuberia.

Forma de uso:

$ capit96 −e usuario −−> hacemos una llamada

$ caipt96 −r usuario −−> contestamos a una llamada */

```
#include <stdio.h>

#include <string.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <utmp.h>

#define MAX 256

#define EQ(str1,str2) (strcmp((str1,str2))==0)

int fifo_12,fifo_21;

char nombre_fifo_12[MAX],nombre_fifo_21[MAX];

char mensaje[MAX];

int estado;

main (argc,argv)

int argc;

char*argv[];

{

int tty;
```

```
enum{LLAMAR,RECIBIR} tipo_llamada;
char terminal [MAX],*cadena,*lognamr,*getenv(),*leer_cadena();
struct utmp *utmp,*getutent();
void fin_de_transmisi\n();
if(argc !=3 || (!EQ(argv[1],"−e") && !EQ(argv[1],"−r")))
{
fprinff(stderr,"Forma de uso: %s−el−r usuario\n*,argv[0]);
exit(−1);
}
else if(EQ(argv[1],"−e"))
tipo_llamada=LLAMAR;
else
tipo_llamada=RECIBIR;
while((utmp=getutent())!=NULL&&
strncmp(utmp−>ut_user,argv[2],8 !=0);
if(utmp==NULL)
{
prinff("EL USUARIO %s NO ESTA EN SESION.\n",argv[2]);
exit(0);
}
logname=getenv("LOGNAME");
sprinff(nombre_fifo_12,"/usr/tmp/%s_%s",logname,argv[2]);
sprinff(nombre−fifo−21,"/usr/tmp/%s_%s",argv[2],logname);
if(tipo_llamada==LLAMAR)
{
sprinff(terminal,"/dev/%s",utmp−>ut_line);
```

```
if((tty=open(terminal,O_WRONLY))==−1)
{
perror(terminal);
exit(−1);
}
sprinff(mensaje,"\n\tLLAMADA PROCEDENTE DEL USUARIO %s\07\n\
RESPONDER ESCRIBIENDO:llamar −r %s\n\n",logname,logname);
write(tty,mensaje,strlen(mensaje)+1);
close(tty);
unlink(nombre_fifo_12);
unlink(nombre_fifo_21);
if(mknod(nombre_fifo_12,S_IFIFO | 0666,0)==−1||
mknod(nombre_fifo_21,S_IFIFO | 0666,0)==−1)
{
perror(argv[0]);
exit(−1);
}
}
if((fifo_21=open(nombre_fifo_21,O_RDONLYIO_NDELAY))==−1 ||
(fifo_12=open(nombre_fifo_12,O_WRONLY))==−1)
{
if(fifo_12==−1) perror(nombre_fifo_12);
else perror(nombre_fifo_21);
exit(−1);
}
if(tipo_llamada==LLAMAR) prinff("LLAMADA ATENDIDA.\07\n");
```

```
estado=fcntl(0,F_GETFL,0);
if(fcntl(0,F_SETFL,estado | O−NDELAY)==−1) perror("fcntl");
signal(SIGINT,fin_de_transmisi\n);
prinff("<==",fflush(stdout);
do
{
if((cadena=leer_cadena())!=NULL)
{
if(cadena[0]!=0)
{ strcpy(mensaje,cadena);
write(fifo_12,mensaje,strlen(cadena)+1);}
prinff("<==",fflush(stdout);
}
if(read(fifo_21,mensaje,MAX)>0)
{ prinff("==>%s\n",mensaje);
prinff("<==",fflush(stdout); }
sleep(1);
}while(!EQ(mensaje,"corto"));
fin_de_transmisi\n();
}
void fin_de_transmisi\n(sig)
int sig;
{
fcntl(0,F_SETFL,estado);
sprinff(mensaje,"corto");
write(fifo_12,mensaje,strlen(mensaje)+1);
```

```
prinff("FIN DE TRANSMISI[N.\n");
close(fifo_12);
close(fifo_21);
fflush(stdin);
exit(0);
}
char *leer_cadena()
{
#define CR 10
static char cadena[MAX];
char cadena_aux[MAX];
static int primera_vez=1;
int nbytes,i;
if(primera_vez)
{
cadena[0]=0;
−−primera_vez;
}
if((nbytes=read(0,cadena_aux,MAX))==0)
return(NULL);
for(i=0;i<nbytes;i++)
if(cadena_aux[i]==CR)
{
cadena_aux[i]=0;
strcat(cadena,cadena_aux);
++primera_vez;
```

```
return(cadena);

}

cadena_aux[i]=0;

strcat(cadena,cadena_aux);

return(NULL);

}
```

/* Este programa sincroniza dos procesos para que se vayan ejecutando de forma alternativa por medio de dos sem<foros por lo menos. Uno de los sem<foros lo usa el padre para darle paso al hijo, y el otro lo utiliza el hijo para darle paso al padre. */

```
#include <stdio.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#define SEM_HIJO 0

#define SEM_PADRE 1

main(argc,argv)

int argc;

char*argv[];

{

int i=10,semid,pid;

struct sembuf operacion;

key_llave;

llave=ftok(argv[0],'k');

if((semid=semget(llave,2,IPC_CREAT | 0600))==-1)

{

perror("semget");
```

```
exit(−1);
}
semctl(semid,SEM_HIJO,SETVAL,09;
semctl(semid,SEM_PADRE,SETVAL,1);
if((pid=fork())==−1)
{
perror("fork");
exit(−1);
}
else if(pid==0)
{
while(i)
{
operacion.sem_num=SEM_HIJO;
operacion.sem_op=1;
operacion_flg=0;
semop(semid,&operacion,1);
printf("PROCESO HIJO:%d\n",i−−);
operacion.sem_num=SEM_PADRE;
operacion.sem_op=1;
semop(semid,&operacion,1);
}
semctl(semid,0,IPC_RMID,0);
}
else
{
```

```
operacion.sem_flg=0;

while(i)

{

operacion.sem_num=SEM_PADRE;

operacion.sem_op=−1;

semop(semid,&operacion,1);

printf("PROCESO PADRE:%d\n",i−−);

operacion.sem_num=SEM_HIJO;

operacion.sem_op=1;

semop(semid,&operacion,1);

}

semctl(semid,0,IPC_RMID,0);

}

}
```

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* MEMORIA COMPARTIDA

Este programa resuelve un algoritmo para multiplicar matrices. El programa principal se encarga de leer dos matrices y comprueba si se pueden multiplicar. Posteriormente arrancaran todos los procesos. Cada proceso se va a ocupar de generar una fila de la matriz producto mientras queden filas por generar. Para controlar la fila que debe generar cada proceso utilizamos un sem<foro que se inicializa con el total de filas de la matriz producto y se va decrementando por cada fila generada. El principal se queda esperando a que los dem<s terminen para dar el resultado. */

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

typedef struct
```

```
{
int shmid;
int filas,columnas;
float **coef;
}matriz;
matriz *crear_matriz(filas,columnas)
int filas,columnas;
{
int shmid,i;
matriz *m;
shmid=shmget(IPC_PRIVATE, sizeof(matriz)+filas* sizeof(float*) + filas*columnas*sizeof(float),
IPC_CREAT | 0600);
if(shmid==−1)
{
perror("crear_matriz(shmdt)");
exit(−1);
}
if((m=(matriz*)shmat(shmid,0,0))==(matriz*)−1)
{
perror("crear_matriz(shmdt)");
exit(−1);
}
m−>shmid=shmid;
m−>filas=filas;
m−>columnas=columnas;
m−>coef=(float**)&m−>coef+sizeof(float**);
for(i=0;i<filas;i++)
```

```
m−>coef[i]=(float*)&m−>coef[filas]+i*columnas*sizeof(float);
return m;
}
matriz *leer_matriz()
{
int filas,columnas,i,j;
matriz * m;
printf("Filas:");
scanf("%d",&filas);
printf("Columnas:");
scanf("%d",&columnas);
m=crear_matriz(filas,columnas);
for(i=0;i<filas;i++)
for(j=0;j<columnas;j++)
canf("%f",&m−>coef[i][j]);
return m;
}
matriz *multiplicar_matrices(a,b,numproc)
matriz*a,*b;
int numproc;
{
int p,semid,estado,i,j,k;;
matriz *c;
if(a−>columnas!=b−>filas) return NULL;
c=crear_matriz(a−>filas,b−>columnas);
semid=semget(IPC_PRIVATE,2,IPC_CREAT | 0600);
```

```
if(semid==−1)
{
perror("multiplicar_matrices(semget)");
exit(−1);
}
semctl(semid,0,SETVAL,1);
semctl(semid,1,SETVAL,c−>filas+1);
for(p=0;p<numproc;p++)
{
if(fork()==0)
{
struct sembuf operacion;
operacion.sem_flg=SEM_UNDO;
while(1)
{
operacion.sem_num=0;
operacion.sem_op=−1;
semop(semid,&operacion,1);
i=semctl(semid,1,GETVAL,0);
if(i>0)
{
semctl(semid,1,SETVAL,−−i);
operacion.sem_num=0;
operacion.sem_op=1;
semop(semid,&operacion,1);
}
```

```
else exit(0);
for(j=0;j<c−>columnas;j++)
{
c−>coef[i][j]=0;
for(k=0;k<a−>columnas;k++)
c−>coef[i][j]+=a−>coef[i][k]*b−>coef[k][j];
} } }
}
for(p=0;p<numproc;p++)
wait(&estado);
semctl(semid,0,IPC_RMID,0);
return c;
}
destruir_matriz(m)
matriz *m;
{
shmctl(m−>shmid,IPC_RMID,0);
}
imprimir_matriz(m)
matriz *m;
{
int i,j;
for(i=0;i<m−>filas;i++)
{
for(j=0;j<m−>columnas;j++)
printf("%g",m−>coef[i][j]);
```

```
printf("\n"); }
}
main(argc,argv)
int argc;
char *argv[];
{
int numproc;
matriz *a,*b,*c;
if(argc!=2) numproc=2;
else numproc=atoi(argv[1])+1;
a=leer_matriz();
b=leer_matriz();
c=multiplicar_matrices(a,b,numproc);
if(c!=NULL) imprimir_matriz(c);
else fprintf(stderr,"Las matrices no se pueden multiplicar.");
destruir_matriz(a);
destruir_matriz(b);
destruir_matriz(c);
}
```

```
/* censo.h
El fichero de cabecera para los clientes y el siguiente programa gestor de una base de datos centralizada */
#ifndef _CENSO_
#define _CENSO_
struct persona{
char nombre[61];
```

```
char direccion[121];

char telefono[11];

};

struct mensaje{

long pid;

int orden;

union

{ struct persona persona;

} datos;

};

#define LONGITUD (sizeof(struct mensaje)−sizeof(long))

#define LISTAR 1

#define ANNADIR 2

#define FIN 3

#define ERROR 4

#define FICHERO_LLAVE "censo.h"

#define CLAVE_CLIENTE_GESTOR 'K'

#define CLAVE_GESTOR_CLIENTE 'L'

#endif
```

−−−−−−−−−−−−−−−−−−−−−−−−−

/* Este programa se va a comportar como un gestor de la BD de

personas. Cada registro consta de:

Nombre:

Direccion:

Tlf:

Para comunicar este programa con el cliente usaremos una cola de mensajes. El tipo de cada cola coincide con el PID del proceso cliente. Cuando varios clientes soliciten servicio del gestor se podra identificar a que

proceso pertenece cada mensaje. Los mensajes que vayan desde el gestor hacia el cliente se envian a travJs de otra cola. */

/* Programa Gestor de la Base de Datos */

```
#include <stdio.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include "censo.h"

main(argc,argv)

int argc;

char *argv[];

{

int cola_cg,cola_gc;

struct mensaje mensaje;

key_t llave;

FILE*pf;

if(argc!=2)

{

fprintf(stderr,"Forma de uso:%s fichero.\n",argv[0]);

exit(−1);

}

llave=ftok(FICHERO_LLAVE,CLAVE_CLIENTE−GESTOR);

if((cola_cg=msgget(llave,IPC_CREAT | 0666))==−1)

{

perror("msget");

exit(−1);

}
```

```
llave=ftok(FICHERO_LLAVE,CLAVE_GESTOR_CLIENTE);
if(cola_gc=msgget(llave,IPC_CREAT | 0666))==−1)
{
perror("msget");
exit(−1);
}
while(1)
{
msgrcv(cola_cg,&mensaje,LONGITUD,0,0);
switch(mensaje.orden)
{
case LISTAR:
if((pf=fopen(argv[1],"r"))==NULL)
{
mensaje.orden=ERROR;
msgsnd(cola_gc,&mensaje,LONGITUD,0);
break;
}
while(fread(&mensaje.datos.persona,sizeof(struct
persona),1,pf)==1)
msgsnd(cola_gc,&mensaje;LONGITUD,0);
fclose(pf);
mensaje.orden=FIN;
msgsnd(cola_gc,&mensaje;LONGITUD,0);
break;
case ANNADIR:
```

```
if((pf=fopen(argv[1],"a"))==NULL)
{
mensaje.orden=ERROR;
msgsnd(cola_gc,&mensaje,LONGITUD,0);
break;
}
fwrite(&mensaje.datos.persona,sizeof(struct persona),1,pf);
fclose(pf)
mensaje.orden=FIN;
msgsnd(cola_gc,&mensaje,LONGITUD,0);
break;
case FIN:
msgctl(cola_cg,IPC_RMID);
msgctl(cola_gc,IPC_RMID);
pritf("Programa gestor parado a petici\n de un cliente.\n");
exit(0);
default:
mensaje.orden=ERROR;
msgsnd(cola_gc,&mensaje;LONGITUD,0):
}
}
}
```

```
/* Cliente de la BD. Puede realizar dos tipos de operaciones:
− Visualizar el contenido de la BD
− AZadir nuevos registros */
```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include "censo.h"
main(argc,argv)
int argc;
char*argv[];
{
int cola_cg,cola_gc,pid;
char opci\n;
struct mensaje mensaje;
key_t llave;
enum {NO,SI} recibir=NO;
llave=ftok(FICHERO_LLAVE,CLAVE_CLIENTE_GESTOR);
if((cola_cg=msgget(llave,0666))==−1)
{
perror("msget");
exit(−1);
}
llave=ftok(FICHERO_LLAVE,CLAVE_GESTOR_CLIENTE);
if((cola_gc=msgget(llave,0666))==−1)
{
perror("msget");
exit(−1);
}
```

```
mensaje.pid=pid=getpid();
while(1)
{
printf("Orden:");
fflush(stdout);
opcion=getchar();
fflush(stdin);
switch(opcion)
{
case '|':
recibir=SI;
mensaje.orden=LISTAR;
msgsnd(cola_cg,&mensaje,LONGITUD,0);
break;
case 'a':
recibir=SI;
printf("\tNombre:");
gets(mensaje.datos.persona.nombre);
printf("\tDirecci\n:");
gets(mensaje.datos.persona.direccion);
printf("\tTelefono:");
gets(mensaje.datos.persona.telefono);
mensaje.orden=ANNADIR;
msgsnd(cola_cg,&mensaje,LONGITUD,0);
break;
case 'f':
```

```
recibir=NO;
mensaje.orden=FIN;
msgsnd(cola_cg,&mensaje,LONGITUD,0);
break;
case 's':
exit(0);
default:
recibir=NO;
printf("Opci\n[%c]err\nea.\n",opcion);
printf("Opciones disponibles:\n");
printf("\ta−aZadir registros.\n");
printf("\tf−para el programa gestor.\n");
printf("\tl−visualizar todos los registros.\n");
printf("\ts−para el programa cliente.\n");
}
if(recibir==SI)
do
{
msgrcv(cola_gc,&mensaje,LONGITUD,pid,0);
switch(mensaje.orden)
{
case FIN:
break;
case LISTAR:
printf("\n\tNombre:%s\n",mensaje.datos.persona.nombre);
printf("\tDirecci\n:%s\n",mensaje.datos.persona.direccion);
```

```
printf("\tTelJfono:%s\n",mensaje.datos.persona.telefono);
break;
case ERROR:
printf("Mensaje de error recibido.\n");
break;
default:
printf("Tipo de mensaje desconocido[%d]\n",
mensaje.orden);
}
}while(mensaje.orden!=FIN && mensaje.orden!=ERROR);
}
}
```

```
/* cat f1 f2 f3 | pr | wc −lc */
# include <stdio.h>
main()
{
int p[2];
int n,q;
if (pipe(p) == −1)
{ fprintf(stderr, "error primer pipe \n");
exit(1); }
if ((n = fork()) == −1 )
{ fprintf (stderr, "error de fork \n");
exit(2); }
if (!n)
```

```
{
close(1);
dup(p[1]);
close(p[1]);
close(p[0]);
execlp ("cat","cat","f1","f2","f3",NULL);
fprintf ("error de exec cat \n");
exit(3);
}
close (p[1]);
q = p[0];
if (pipe(p) == −1 )
{ fprintf (stderr, "error del segundo pipe \n");
exit(4); }
if ((n=fork()) == −1)
{ fprintf (stderr, "error del fork \n");
exit(5); }
if(!n)
{ close(0);
dup(q);
close(q);
close(1);
dup (p[1]);
close(p[1]);
close(p[0]);
execlp ("pr","pr",NULL);
```

```
exit(6); }

close(q);

close(p[1]);

close(0);

dup(p[0]);

close(p[0]);

execlp ("wc","wc","−lc",NULL);

fprintf(stderr, "error de exec wc \n");

}

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Uso de getpid(),getppid(),getpgrp(),getuid(),geteuid() */

#include <stdio.h>

main(argc, argv)

int argc;

char *argv[];

{

int rc=0, estado;

if (argc!=1) {

fprintf(stderr,"uso: fork1\n");

rc=1;

} else if (fork()) {

wait (&estado);

printf ("PADRE pid=%8d ppid=%8d id−grupo=%8d ur=%8d
ue=%8d\n",getpid(),getppid(),getpgrp(),getuid(),geteuid());

printf ("El codigo de retorno del hijo es: %d\n",estado);

} else

printf ("HIJO pid=%8d ppid=%8d id−grupo=%8d ur=%8d
```

```
ue=%8d\n",getpid(),getppid(),getpgrp(),getuid(),geteuid());
exit (rc);
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Uso de pipe() */
#include <stdio.h>
#include <fcntl.h>
main(argc, argv, envp)
int argc;
char *argv[];
char *envp;
{
int p[2],d1,d2,n;
char ch[10];
pipe(p);
d1=open("f",O_RDWR|O_CREAT|O_TRUNC,0666);
if (fork()!=0) {
sleep(1);
printf ("Lanzamiento del proceso 1\n");
d2=open("f",O_RDWR,0);
write (p[1],"abcde",5);
sleep(5);
write (p[1],"fghij",5);
n=read (p[0],ch,10);
printf ("Proceso 1: n=%d\n",n);
write (d2,ch,n);
```

```
sleep(5);
read (d1,ch,2);
printf ("Proceso 1: %c %c\n",ch[0],ch[1]);
write (d1,"AB",2);
sleep(5);
read (d2,ch,1);
printf ("Proceso 1: %c\n",ch[0]);
read (d1,ch,1);
printf ("Proceso 1: %c\n",ch[0]);
_exit(0);
}else{
printf ("Lanzamiento del proceso 2\n");
d2=open("f",O_RDWR,0);
sleep(2);
n=read (p[0],ch,10);
printf ("Proceso 2: n=%d\n",n);
write (d2,ch,n);
sleep(5);
read (d1,ch,2);
printf ("Proceso 2: %c %c\n",ch[0],ch[1]);
sleep(5);
write (d2,"01",2);
_exit(0);
}
}
```

```
/* Uso de signal () */

#include <stdio.h>

main(argc, argv)

int argc;

char *argv[];

{

int child;

if (argc!=1) {

fprintf(stderr,"uso: fork5\n");

exit(1);

} else if ((child=fork())==0) {

printf ("HIJO pid=%8d ppid=%8d\n",getpid(),getppid());

signal(SIGINT,catcher)

pause(); }

printf ("PADRE pid=%8d ppid=%8d\n",getpid(),getppid());

printf ("HIJO pid=%8d ppid=%8d\n",child,getppid());

exit (child);

}

int catcher(signo)

int signo;

{

printf ("Hola soy yo de nuevo.\n");

printf ("Atrapada la seZal ndmero %d\n",signo);

fflush (stdout);

}

_____
```

```
/* Uso de getxxid () */

#include "/usr/include/string.h"

#include "/usr/include/stdio.h"

main(argc, argv, envp)

int argc;

char *argv[];

char *envp[];

{

int ret;

if (!strcmp(argv[1],"−p") || !strcmp(argv[1],"−P"))

{

printf("mi ID de proceso primario : %d\n",getppid());

printf("mi ID de grupo es : %d\n",getpgrp());

}

else if (!strcmp(argv[1],"−g") || !strcmp(argv[1],"−G"))

{

printf ("mi ID de grupos real de procesos es: %d\n", getgid());

printf ("mi ID de grupo efectivo de procesos es: %d\n", getegid());

}

else if (!strcmp(argv[1],"−u") || !strcmp(argv[1],"−U"))

{

printf ("Mi ID de usuario real es : %d\n", getuid());

printf ("Mi ID de usuario efectivo es : %d\n", geteuid());

}

else

printf(" %s ES UNA OPCION INVALIDA ... PRUEBA DE NUEVO\n Opciones: −p −g −u\n",argv[1]);
```

```
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* ejemplo de llamadas al sistema: msgctl, msgsnd, msgrcv */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define DIM 100
char txt[] ="Mensaje";
struct mansae {
int msg_id;
char texto[DIM] ;} mensaje;
main()
{
int i,key,ppid,estatus;
struct mansae *pmensaje;
struct msgqid_ds *buf;
pmensaje=&mensaje;
if ((key=msgget(IPC_PRIVATE,IPC_CREAT|0666)) ==−1)
errorSistema("msgget");
system("ipcs −q");
sleep(2);
if ((ppid=getpid())==−1)
errorSistema("getpid");
switch(fork())
{ case −1: errorSistema("fork");
```

```
case 0: sleep(1);
if(msgrcv(key,mensaje,DIM,ppid,IPC_NOWAIT)==−1)
errorSistema("msgrcv");
printf("Mensaje del Hijo\nHe recibido:%s n",pmensaje−texto);
exit(0);
default: sprintf(pmensaje−>texto,"%s \n",txt);
pmensaje−>msg_id=ppid;
if(msgsnd(key,pmensaje,DIM,IPC_NOWAIT)==−1)
errorSistema("msgsnd");
i=wait(&estatus);
msgctl(key,IPC_RMID,buf);
system("ipcs −q");
sleep(1);
printf("Mensaje del Padre: Continuo \n");
return(0); }
}
−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de llamadas al sistema: shmget, shmat, shmdt*/
#include <stdio.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMKEY 0X100
char *region;
int shmid;
```

```
main()
{
int i, id_hijo;
char miregion[10],*rp;
char *shmat();
if ((shmid=shmget((key_t)(SHMKEY),256,IPC_CREAT|0664))>0)
{ if((shmid=shmget((key_t)(SHMKEY),0,IPC_EXCL|0664))>0)
{ perror("shmget"); exit(1); } }
switch(fork())
{ case −1: perror("fork"); return (2);
case 0: signal(SIGUSR1,enmarcha);
signal(SIGUSR2,termina);
pause();
if((region=shmat(shmid,region,SHM_RND|0664))<(char *)0)
{ perror("Hijo:shmat"); return(3); }
strncpy(miregion,region,10);
for(;;)
{ for(i=0;i<10;i++)
{ if(region[i]!=miregion[i])
{ printf("\n\n OJO hay CAMBIOS !!!\n");
printf(" DE: %s \n",miregion);
printf(" A: %s \n",region);
strncpy(miregion,region,10); } } }
default: break;
}
if((region=shmat(shmid,region,SHM_RND|0664))<(char *)0)
```

```
{ perror("Padre:shmat"); return(3); }
for(i=0;i<10;i++) region[i]='−';
region[9]='\0';
sleep(1);
kill(id_hijo,SIGUSR1);
region[1]='a';
sleep(1);
region[3]='b';
sleep(3);
region[5]='c';
sleep(2);
region[7]='d';
sleep(2);
region[8]='e';
sleep(1);
kill(id_hijo,SIRUSR2);
termina();
}
termina()
{
shmdt(region);
shmctl(shmid,IPC_RMID,NULL);
exit(0);
}
enmarcha()
{
```

```
printf("Hijo: (( QuJ ganas tenRa de empezar !!!\n");

return;

}

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Uso de fork() */

#include <stdio.h>

main()

{

int status, frkpid, chldpd;

printf("Inicio del proceso principal...\n");

if ((frkpid = fork()) == 0 )

{ printf("hijo: mi id de proceso es %d\n",getpid());

printf("hijo: mi id de proceso padre es %d\n",getppid());

printf("hijo: mi id de grupo de procesos es %d\n",getpgrp());

printf("fin del proceso hijo...\n");

exit(0); }

chldpd=wait(&status);

printf("padre: el id de mi proceso hijo es %d\n", chldpd);

printf("padre: el id de mi proceso es %d\n", getpid());

printf("padre: el id de mi proceso padre es %d\n", getppid());

printf("padre: el id de mi grupo de procesos es %d\n",getpgrp());

printf("fin del proceso padre...\n");

}

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

/* Ejemplo de llamadas al sistema: semget y semop

Ejemlpo de producci\n y consumo. El proceso Padre produce dos elementos y 2 procesos hijos los consumen,
si hay suficientes. Una mejora a introducir en este ejemplo es el liquidar el sem<foro al terminar. Este ejemplo
```

```
es poco correcto, pues se fia del tiempo para una ejecuci\n adecuada. */
#include <stdio.h>
#include <sys/types.h>
#include <sys/sem.h>
#include <sys/ipc.h>
mainz()
{ int semid;
struct sembuf mibuf;
if ((semid=semget(IPC_PRIVATE,1,0666)) ==−1)
{ perror("semget"); exit(1); }
switch (fork())
{ case −1: { perror("fork");exit(2); }
case 0: /* Hijo A */
sleep(3);
printf("Hijo A: Quiero 50 unidades \n");
mibuf.sem_num=0;
mibuf.sem_op=−50;
mibuf.sem_flg=0;
if ( semop(semid,&mibuf,1)==−1)
{ perror("semop"); exit(3); }
printf("Hijo A: Recibidas las 50 unidades\n");
sleep(1);
printf("Hijo A: Ahora qiuero 80 unidades m<s \n");
mibuf.sem_op=−80;
if (semop(semid,&mibuf,1)==−1)
{ perror("semop"); exit(4); }
```

```
printf("Hijo A: Recibidas las 80 unidades \n");
sleep(1);
printf("Hijo A: Termino por hoy\n");
exit(0);
default :
switch(fork())
{ case −1: { perror("fork"); exit(2); }
case 0: /* Hijo B */
sleep(3);
printf("Hijo B: Quiero 20 unidades \n");
mibuf.sem_num=0;
mibuf.sem_op=−20;
mibuf.sem_flg=0;
if ( semop(semid,&mibuf,1)==−1)
{ perror("semop"); exit(5); }
printf("Hijo B: Recibidas las 20 unidades\n");
sleep(1);
printf("Hijo B: Ahora qiuero 100 unidades m<s \n");
mibuf.sem_op=−100;
if (semop(semid,&mibuf,1)==−1)
{ perror("semop"); exit(5); }
printf("Hijo B: Recibidas las 100 unidades \n");
sleep(1);
printf("Hijo B: Termino por hoy\n");
exit(0);
default :
```

```
printf("Padre: Intento producir 100 unidadesn");
mibuf.sem_num=0;
mibuf.sem_op=100;
mibuf.sem_flg=0;
if ( semop(semid,&mibuf,1)==−1)
{ perror("semop"); exit(6); }
printf("Padre : Ya he producido 100\n");
sleep(5);
printf("Padre : Ahora voy a hacer 70 m<s\n");
mibuf.sem_op=70;
if ( semop(semid,&mibuf,1)==−1)
{ perror("semop"); exit(7); }
printf("Padre : Ya he producido 70\n");
sleep(5);
printf("Padre : Y por fin voy a hacer 80\n");
mibuf.sem_op=80;
if ( semop(semid,&mibuf,1)==−1)
{ perror("semop"); exit(8); }
printf("Padre : Ya est<n las 80\n");
sleep(5);
printf("Padre : Cierro el taller ahora mismo\n");
exit(0); } }
}
```

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

```
/* Uso de signal() */
#include <stdio.h>
```

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

#define ERROR (−1)

static int caught=0;

main(argc, argv)

int argc;

char *argv[];

{

int (*oldint)(), rc=0, catcher();

if (argc!=1) {

fprintf(stderr,"uso: signal1\n");

rc=1;

} else if ((oldint=signal(SIGINT,catcher))==(int (*)()) ERROR){

perror(argv[0]);

rc=1;

} else {

for (caught=0;!caught;)

printf ("No hay seZal todavia\n");

}

exit (rc);

}

int catcher(signo)

int signo;

{

caught=1;

printf ("Atrapada la seZal ndmero %d\n",signo);

fflush (stdout);
```

```
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Uso de la llamada del sistema gettimer() */
#include <stdio.h>
#include <sys/time.h>
main()
{
long secs;
int ret;
int retc;
struct timestruc_t *tp;
tp=(struct timestruc_t *)malloc(sizeof(struct timestruc_t));
printf("Utilizando la llamada del sistema gettimer... \n");
ret=gettimer(TIMEOFDAY,tp);
printf("El tiempo transcurrido es %d\n",tp−>tv_sec);
printf("Suspendiendo la ejecucion por 5 segundos...\n");
retc=sleep(5);
printf("Fin de la suspension...\n");
printf("Se utliza la llamada al sistema time... \n");
secs=time(0);
printf("El tiempo transcurrido es %d.\n",secs);
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
#include <stdio.h>
#include <string.h>
#define MAXARG 20
```

```
#define CIERTO 1
#define FALSO 0
#define PS1 "Orden :"
static void ejecuta (int argc, char *argv[]);
static int getargs (int *argcp, char *argv[], int max);
void main(void)
{ int argc;
char *argv [MAXARG+1];
while (CIERTO) {
printf("%s",PS1);
if (!getargs(&argc,argv,MAXARG) || argc==0)
continue; /* Volver a leer una orden */
ejecuta (argc,argv); }
printf("FIN DE EJECUCION \N"); }
static int getargs (int *argcp, char *argv[], int max)
{ static char cmd [100];
char *cmdp;
int i;
if (gets(cmd)==NULL) return (FALSO);
cmdp=cmd;
for (i=0;i<=max;i++)
{ if((argv[1]=strtok(cmdp, " \t"))==NULL) break;
cmdp=NULL; }
if (i>max) {
printf("Demasiados argumentos \n");
return(FALSO); }
```

```
*argcp=i;
return(CIERTO); }
static void ejecuta (int argc, char *argv[])
{ execvp (argv[0], argc);
printf("No puedo ejecutarlo \n");
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Borrado de un fichero temporal cuando el programa es
abortado con un CTRL−C (SeZal SIGINT) */
#define ERROR −1
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
main()
{ int i, accion();
if (creat("tmp",S_IREAD+S_WRITE)==ERROR)
{ perror("create"); exit(1); }
if (signal(SIGINT,SIG_IGN)!=SIG_IGN)
signal(SIGINT,accion);
for (i=1; i<100000; i++)
printf("%d \n",i);
unlink("tmp"); /* por si el bucle termina sin haber seZal */
return (0);
}
int accion()
```

```
{
if (unlink("tmp")==ERROR)
{ perror("unlink");
exit(1); }
else exit(0);
}
−−−−−−−−−−−−−−−−−−−−−−−−−−−
/* Ejemplo de llamadas al sistema: kill, pause, signal */
#define ERROR −1
#include <stdio.h>
#include <signal.h>
main()
{ int estatus,pid,accion();
switch(pid=fork())
{ case ERROR: perror("fork");
exit(1);
case 0:
signal (SIGHUP,accion);
printf("Mensaje de HIJO: Espero seZal\n");
pause();
break;
default:
sleep(2);
printf("Mensaje de PADRE: Mando seZal\n");
kill(pid,SIGHUP);
wait(estatus);
```

```
printf ("Mensaje de PADRE: HIJO ha terminado\n"); }

return (0);

}

int accion()

{

printf("Mensaje de HIJO: SeZal recibida. Continuo\n");

}
```

53