LABORATORIO DE SISTEMAS TELEMÁTICOS

PRÁCTICA PREVIA

#include<stdio.h>#include<fcntl.h>#define PERMS 0644#define BUFSZ 512

main(argc,argv){

int argc;char *argv[];int fildes1,fildes2,n_read,n_write;char datos[BUFSZ];if (argc!=3) { printf("Error \n Sintaxis: %s fichero_fnt fichero_des\n",

argv[0]);

exit(1);

}

if ((fildes1=open(argv[1],O_RDONLY))==−1)

{

perror("No se pudo abrir el fichero fuente");

exit(1);

}

if ((fildes2=open(argv[2],O_CREAT|O_TRUNC|O_RDWR,PERMS))==−1)

{

perror("No se pudo abrir el fichero destino");

exit(1);

}

do

{

if ((n_read=read(fildes1,datos,BUFSZ))==−1)

{

perror("Error de lectura");

exit(1);

}

```
if ((n_write=write(fildes2,datos,n_read))==−1)

{

perror("Error de escritura");

exit(1);

}

}while (n_read!=0);

close(fildes1);

close(fildes2);

}
```

Cuestiones:

- Plantearía problemas el hecho de abrir el fichero fuente en modo lectura escritura. ¿Qué modificaciones habría que realizar en el programa para abrir el fichero origen en modo lectura escritura?

En principio no representaría ningún problema. Debería sustituirse la línea:

if ((fildes1=open(argv[1],O_RDONLY))==−1)

por −> if ((fildes1=open(argv[1],O_RDWR))==−1)

- Los flags O_TRUNC y O_CREAT empalados en el programa para qué se emplean. Indique donde se encuentra definido el valor de estas constantes.

O_CREAT: Si el fichero que queremos abrir ya existe, este indicador no tiene efecto, excepto en lo que indicará para el indicador O_EXCL. El fichero es creado en caso de que no exista y se creará con los permisos indicados en el parámetro *mode*.

O_TRUC: Si el fichero existe, trunca su longitud a cero bytes, incluso si el fichero se abre para leer.

Estas opciones están incluidas en una máscara de bits que le indica al núcleo el modo en que queremos que se abra el fichero.

- En la escritura en el fichero destino, se manda escribir n_read datos, se podría sustituir n_read por la constante BUBSZ.

No, porque pueda ocurrir el caso en que se lean menos bytes que BUFSZ, por ejemplo al final del fichero.

- Cómo podríamos obtener el código fuente en ensamblador del programa copiar.

Añadiendo la opción `−S' al compilador, el cual funciona ahora como un ensamblador, dando a la salida un fichero `.s', que contiene el código fuente en ensamblador.

PRÁCTICA 1

```
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

PRACTICA 1 PRACTICA 1 PRACTICA 1 PRACTICA 1 PRACTICA 1

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

El programa funciona como la orden 'ls' del shell de UNIX de forma reducida. Parámetros en la línea de
comandos:

* directorio: se puede especificar el directorio del que se quiere listar la información.

* −l: muestra información adicional.

* −i: muestra el i_node.

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <dirent.h>

#include <pwd.h>

#include <grp.h>

#include <sys/types.h>

#include <sys/stat.h>

#define SI 1 /* Constantes booleanas */

#define NO 0

typedef int BOOL; /* Definicion del tipo BOOL */

char permisos []={'x','w','r'};/*Definicion de los caracte−

res de permisos */

/* Función que mostrara los datos */int mostrar(struct stat *datos, char *nombre, BOOL sil, BOOL

sii);/* Programa principal */

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

```
{
DIR *directorio; /*Puntero a la ruta del directorio a abrir*/
BOOL opl=NO; /* Información sobre si se opcionó '−l' */
BOOL opi=NO; /* Información sobre si se opcionó '−i' */
int con; /* Contador para extraer los parámetros */
int opdir=0; /*Posición del nombre del directorio en la LC*/
struct dirent *entra_dir; /* Entrada del directorio leído */
struct stat datos_fic; /* Información sobre el estado del
fichero */
/* Comprobación de la línea de comandos */
if (argc>4)
{
printf("Error \n Sintaxis: %s [Directorio] [−l] [−i]",
argv[0]);
exit(−1);
}
/* Miro las opciones introducidas en la línea de comandos */
for (con=1;con<argc;con++)
{
if (strcmp(argv[con],"−l")==0) opl=SI;
else if (strcmp(argv[con],"−i")==0) opi=SI;
else opdir=con;
}
/* Si se ha especificado el directorio se abre, sino, se abre el directorio actual */
if (opdir)
{
```

```
if ((directorio=opendir(argv[opdir]))==NULL)
{
printf("Error en la apertura del directorio %s",
argv[opdir]);
exit(−1);
}
}
else
{
if ((directorio=opendir("."))==NULL)
{
printf ("Error en la apertura del directorio actual");
exit(−1);
}
}
/* Extraer todas las entradas del directorio */
while ((entra_dir=readdir(directorio))!=NULL)
{
if (stat(entra_dir−>d_name, &datos_fic)==−1)
{
printf ("Error en la lectura de datos del fichero %s",
entra_dir−>d_name);
exit(−1);
}
/* Muestra de los datos de los ficheros */
if (mostrar(&datos_fic, entra_dir−>d_name, opl, opi)==−1)
```

```
{
printf ("Error en la muestra de los datos");
exit(−1);
}
}
printf("\n"); /* Deja una línea en blanco */
/* Cerramos el directorio */
if(closedir(directorio)==−1)
{
printf ("Error al cerrar el directorio");
exit(−1);
}
}
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
Función de muestra de datos. Parámetros:
* datos: puntero a la estructura que contiene los datos de la entrada del directorio.
* nombre: nombre de la entrada del directorio explorado.
* sil: vale SI si en la línea de comandos esta la opción '−l'
* sii: vale SI si en la línea de comandos esta la opción '−i'
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/
int mostrar (struct stat *datos, char *nombre, BOOL sil, BOOL
sii)
{
struct passwd *propietario; /* Contiene el nombre del
propietario */
struct group *grupo; /* Contiene el nombre del grupo */
```

```
int i; /* Entero para indexar */

/* Miro que información tengo que presentar, según los comandos */

if (sii==SI) printf ("%7d ", datos−>st_ino);

if (sil==NO) printf ("%−12s", nombre);

else

{

/* Obtención del tipo de fichero */

switch(datos−>st_mode & S_IFMT)

{

case S_IFREG: /* Ordinario */

printf ("−");

break;

case S_IFDIR: /* Directorio */

printf ("d");

break;

case S_IFCHR: /* Especial modo carácter */

printf ("c");

break;

case S_IFBLK: /* Especial modo bloque */

printf ("b");

break;

case S_IFIFO: /* FIFO (tuberia) */

printf ("p");

break;

default:

printf(" ");
```

```
}
/* Extracción de los permisos */
for (i=0;i<9;i++)
if (datos−>st_mode & (0400 >> i))
printf("%c",permisos [(8−i)%3]);
else
printf("−");
/* Extracción del numero de enlaces */
printf("%4d ",datos−>st_nlink);
/* Obtención del nombre del propietario */
if ((propietario=getpwuid(datos−>st_uid))==NULL)
return(−1);
/* Obtención del nombre del grupo */
if ((grupo=getgrgid(datos−>st_gid))==NULL)
return(−1);
/* Impresión de los nombres de propietario y grupo */
printf("%s\t%s ",propietario−>pw_name, grupo−>gr_name);
/* Impresión de la longitud de la entrada */
printf("%12d ",datos−>st_size);
/* Obtención de los tiempos asociados */
/* printf("%s", asctime(localtime (&datos−>st_atime)));*/
/* printf("%s", asctime(localtime (&datos−>st_mtime)));*/
/* printf("%s", asctime(localtime (&datos−>st_ctime)));*/
/* Impresión del nombre de la entrada */
printf("%s\n",nombre);
}
```

```
return(0);

}
```

Cuestiones:

- Desarrolle una función uid2nombre() a la cual se le pase como argumento el uid de un usuario y nos devuelva el nombre del usuario si se ejecuta de forma correcta o −1 si se produce algún fallo (p.e. el usuario no existe).

Primero mostraremos un programa que no utiliza llamadas para obtener el nombre.

```
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−Esta es la función uid2nombre() a la cual se le pasa como argumento el uid de un usuario y nos devuelve el nombre de usuario en caso de existir o −1 si se produce algún error (p.e. el usuario no existe).

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

#include <stdio.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

main (int argc, char *argv[])

{

int nbytes,fd_origen; /* Valores de referencia del fichero */

char carac; /* Carácter leído del fichero */

char nombre[16]; /* Tabla donde se guarda el nombre del

usuario*/

char UID[16]; /* Tabla donde se guarda el uid del usuario */

int i=0; /* Variable que indexa las tablas */

int campo=1; /* Variable que indica el campo examinado */

int num=0; /* Variable que indica la longitud del nombre */

int lon=0; /* Variable que indica la longitud del uid */

/* Comprobación de la línea de comandos */

if (argc!=2)
```

```
{
printf("\nError −−−> Sintaxis: %s UID\n\n", argv[0]);
exit(−1);
}
/* Abrimos el fichero 'passwd' en modo lectura */
if ((fd_origen=open("/etc/passwd",O_RDONLY))==−1)
{
printf("No se pudo abrir el fichero /etc/passwd");
exit(−1);
}
/* Leemos del fichero carácter a carácter hasta el final del fichero */
while ((nbytes=read(fd_origen,&carac,1))>0)
{
/* procesamos línea por línea */
if (carac!='\n')
/* procesamos campo por campo */
if (carac!=':')
switch(campo)
{
case 1: /*si estamos en el primer campo,*/
nombre[i]=carac; /* almacenamos el nombre */
i++;
num=i;
break;
case 3: /* si estamos en el tercer campo,*/
UID[i]=carac; /* almacenamos el uid */
```

```
i++;

lon=i;

break;

}

/* Cambiamos de campo */

else

{

campo++;

i=0;

}

/* Fin de línea */

else

{

/* Comprobamos si el uid del usuario procesado es el pedido, y si lo es, lo mostramos por pantalla y terminamos la búsqueda*/

UID[lon]='\0';

if (strcmp(UID,argv[1])==0)

{

printf("\nNOMBRE DEL USUARIO: ");

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

printf("%c",nombre[i]);

printf("\n\n");

return(0);

}

/* Si no lo es, pasamos a la línea siguiente */

else

{
```

```
campo=1;

i=0;

}

}

}

/* No ha aparecido ningún usuario con el uid especificado */

printf("\n\nNO EXITE ESE USUARIO\n\n");

return(−1);

}
```

Este segundo programa, es mucho más sencillo, y realiza la misma función que el anterior, pero con una llamada mediante la función getpwuid, la cual obtiene el nombre del usuario automáticamente.

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

main (int argc, char *argv[]){ struct passwd *propietario;

int x;

x=atoi(argv[1]);

if ((propietario=getpwuid(x))==NULL)

{

printf("\n\nEL USUARIO NO EXISTE\n\n");

return(−1);

}

printf("\n\n%s\n\n",propietario−>pw_name);

return(0);

}
```

- Desarrolle una función gid2grupo() a la cual se le pase como argumento el gid de un usuario y nos devuelva el nombre del grupo si se ejecuta de forma correcta o −1 si se produce algún fallo (p.e. el grupo no existe).

```
#include <stdio.h>#include <sys/types.h>#include <grp.h>main (int argc, char *argv[])

{
```

```
struct group *grupo;

int x;

printf("\nINTRODUCE EL GID: ");

scanf("%d",&x);

if ((grupo=getgrgid(x))==NULL)

{

printf("\n\nEL GRUPO NO EXISTE\n\n");

return(−1);

}

printf("\n\n%s\n\n",grupo−>gr_name);

return(0);

}
```

- ¿Qué ocurre si dentro de un subdirectorio donde se realiza la búsqueda existe un enlace blando (soft link) del directorio consigo mismo?

Ocurre que nos metemos en un bucle infinito donde se repite el listado del directorio continuamente.

- ¿Qué modificaciones necesitaría el programa para que nos devolviese el total de ficheros cuyo propietario es el especificado como argumento del programa?

Simplemente cada vez que mostráramos la información del fichero, compararíamos el propietario con el argumento dado, incrementando un contador en caso de coincidir. Al final, mostraríamos el contador por pantalla.

PRÁCTICA 2

/*−−−PRACTICA 2 PRACTICA 2 PRACTICA 2 PRACTICA 2 PRACTICA 2

−−−

Este programa esta compuesto de tres procesos:

* P1 realiza la lectura de un fichero introducido en la línea de comandos, y lo transmite carácter a carácter a través de una tuberia.

* P2 recoge la información transmitida por P1, selecciona las mayúsculas, las saca por pantalla y las transmite por otra tuberia.

* P3 recoge la información transmitida por P2, cuenta el numero de caracteres recogidos y los muestra por pantalla

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

#include <stdio.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

main(int argc, char *argv[])

{

char carac; /* carácter del fichero leído */

char c_rec; /* carácter recibido */

char mayus_rec; /* mayúscula recibida */

char FIN='\0'; /* carácter de fin de transmisión */

int nbytes; /* cantidad de datos leídos del fichero */

int fd_origen; /* identificador del fichero leído */

int tuberia1[2]; /* tuberia de transmisión de datos de

P1 a P2 */

int tuberia2[2]; /* tuberia de transmisión de datos de

P2 a P3 */

int pid1,pid2; /*identificadores de los procesos creados*/

int contador=0; /* numero de mayúsculas */

/* Comprobación de la línea de comandos */

if (argc!=2)

{

printf("Error\n Sintaxis: %s fichero\n", argv[0]);

exit(−1);

}
```

```
/* Abrimos la primera tuberia */
if (pipe (tuberia1)==−1)
{
perror("Error al abrir la tuberia");
exit(−1);
}
/* Abrimos el primer proceso hijo */
if ((pid1=fork())==−1)
{
perror("Error al abrir el proceso hijo");
exit(−1);
}
else if (pid1==0)
{
/* Este el primer proceso hijo */
/* Abrimos la segunda tuberia */
if (pipe (tuberia2)==−1)
{
perror("Error al abrir la tuberia");
exit(−1);
}
/* Abrimos el segundo proceso hijo */
if ((pid2=fork())==−1)
{
perror("Error al abrir el proceso hijo");
exit(−1);
```

```
}

else if (pid2==0)

{

/* P3. El segundo proceso hijo */

/* Leemos de la tuberia los datos transmitidos por P2 hasta el fin de transmisión. Sacamos por pantalla los
caracteres, y los contamos */

printf("\nMAYUSCULAS RECIBIDAS: ");

while(read(tuberia2[0],&mayus_rec,1)>0 &&

(mayus_rec!=FIN))

{

printf("%c",mayus_rec);

contador++;

}

/* Sacamos por pantalla el numero de mayúsculas */

printf("\n\nNUMERO DE MAYUSCULAS RECIBIDAS: %d\n\n",

contador);

/* Cerramos las tuberías */

close (tuberia1[0]);

close (tuberia1[1]);

close (tuberia2[0]);

close (tuberia2[1]);

exit(0);

}

else

{

/* P2. El segundo proceso padre */

/* Leemos de la tuberia los datos transmitidos por P1 hasta el fin de transmisión. A continuación
```

```
seleccionamos las mayúsculas, y estas las mostramos en pantalla y las transmitimos a P3 */
printf("\n\nMAYUSCULAS ENVIADAS: ");
while (read(tuberia1[0],&c_rec,1)>0 && (c_rec!=FIN))
{
if ((c_rec>='A')&&(c_rec<='Z'))
{
printf("%c",c_rec);
if (write (tuberia2[1],&c_rec,1)<0)
printf("Error al escribir en la tuberia21");
}
}
/* Mandamos por la tuberia el fin de transmisión */
if (write (tuberia2[1],&FIN,1)<0)
printf("Error al escribir en la tuberia22");
/* Cerramos las tuberías */
close (tuberia1[0]);
close (tuberia1[1]);
close (tuberia2[0]);
close (tuberia2[1]);
exit(0);
}
}
else
{
/* P1. Este es el primer proceso padre */
/*Abrimos el fichero introducido en la línea de comandos*/
```

```
if ((fd_origen=open(argv[1],O_RDONLY))==−1)
{
printf("No se pudo abrir el fichero %s",argv[1]);
exit(−1);
}
/* Leemos el fichero y lo escribimos en la tuberia1 carácter a carácter */
while ((nbytes=read(fd_origen,&carac,1))>0)
{
if (write (tuberia1[1],&carac,1)<0)
printf("error en la transmisión por la tuberia11\n");
}
/* Mandamos por la tuberia el fin de transmisión */
if (write (tuberia1[1],&FIN,1)<0)
printf("error en la transmisión por la tuberia12\n");
/* Cerramos el fichero y las tuberías */
close (fd_origen);
close (tuberia1[0]);
close (tuberia1[1]);
exit(0);
}
}
```

Cuestiones:

- ¿Cuándo terminan de ejecutarse P1, P2 y P3?

P1: Al terminar de escribir en la tubería el fin de mensaje.

P2: Al terminar de escribir en la tubería el fin de mensaje.

P3: Al terminar de escribir las mayúsculas.

- ¿Cuándo un proceso lee de una tubería y nadie ha escrito en ella, qué ocurre?

Ocurre que el proceso que pretende leer se queda dormido esperando leer algo.

- ¿Puede un proceso quedarse bloqueado indefinidamente? ¿Cómo se evita este problema?

Puede ocurrir en el caso anterior, y esto se soluciona (como hemos hecho en la práctica) mandando un *fin de transmisión* para que el receptor no se quede colgado esperando más caracteres.

También se debe hacer una correcta programación para que no haya problemas de lectura−escritura en la programación con tuberías.

<u>PRÁCTICA 3</u>

/*−− PRACTICA 3 PRACTICA 3 PRACTICA 3 PRACTICA 3 PRACTICA 3−−−PROCESO ESCRITOR: Este proceso guarda los caracteres introducidos por el usuario en una zona de memoria compartida con el proceso lector, valiendose para ello de los mecanismos IPC (llaves, semáforos, memoria compartida).

−−*/

#include <stdio.h>

#include <fcntl.h>

#include <sys/ipc.h>

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/stat.h>

#define SMUTEX 0 /* Semáforo de exclusi¢n mutua */

#define SOCUP 1 /* Semáforo que indica las posiciones

ocupadas */#define SVACIO 2/*Semáforo que indica las posiciones vacias*/#define TAM 512 /* Tamaño de la zona de memoria compartida */main(int argc, char *argv[]){ key_t llave; /* Llave de identificaci¢n de los mecanismos

IPC */ int shmid; /* Identifica la zona de memoria compartida */ int semid; /* Identificador del semáforo */ int idfifo; /* Identificador de la tubería */ char *tabla; /* Tabla donde se almacenan los caracteres

escritos */ struct sembuf operacion[3]; /* Estructura de operaciones

```
sobre semáforos */ int i=0; /* Variable que indexa la tabla */ char dato; /* Variable que almacena el dato leido
*/

/* Creamos una llave */

if ((llave=ftok("out.out",'O'))==(key_t)−1)

{

printf("\nError al crear la llave o falta fichero

'out.out' \n"); exit(−1); } /* Creacion de la zona de memoria compartida */

if ((shmid=shmget(llave,TAM,IPC_CREAT|0600))==−1)

{

printf("Error al crear la memoria compartida\n");

exit(−1);

}

/* Atarnos al segmento de memoria compartida */

tabla=shmat(shmid,0,0);

/* Creación de los semáforos e inicialización de los mismos */

if ((semid=semget(llave,3,IPC_CREAT|0600))==−1)

{

printf("Error al crear los semaforos\n");

exit(−1);

}

if (semctl(semid,SMUTEX,SETVAL,1))

{

printf("Error al inicializar el semaforo\n");

exit(−1);

}

if (semctl(semid,SOCUP,SETVAL,0))

{
```

```
printf("Error al inicializar el semaforo\n");
exit(−1);
}
if (semctl(semid,SVACIO,SETVAL,TAM))
{
printf("Error al inicializar el semaforo\n");
exit(−1);
}
/* Abrimos la tubería en modo escritura y escribimos la llave en ella */
if ((idfifo=open("tuberia",O_WRONLY))==−1)
{
printf("\nError al abrir la tuberia\n");
exit(−1);
}
if ((write (idfifo,&llave,sizeof(key_t)))==−1)
{
printf("\nError al escribir en la tuberia\n");
exit(−1);
}
printf("\n−−−− INTRODUCE EL TEXTO HASTA 'Q' −−−−−−\n");
do
{
/* Leemos el dato */
dato=getchar();
/* P sobre los semáforos */
operacion[0].sem_num=SVACIO;
```

```
operacion[0].sem_op=−1;
operacion[0].sem_flg=0;
operacion[1].sem_num=SMUTEX;
operacion[1].sem_op=−1;
semop(semid,operacion,2);
/* Escribimos en la zona de memor¡a compartida */
tabla[i]=dato;
/* V sobre los semáforos */
operacion[0].sem_num=SMUTEX;
operacion[0].sem_op=1;
operacion[1].sem_num=SOCUP;
operacion[1].sem_op=1;
semop(semid,operacion,2);
/* Al llegar al final reinicamos */
(i<TAM)?(i++):(i=0);
} while ((dato!='q')&&(dato!='Q'));
/* Borrado de los semáforos */
shmctl(shmid,IPC_RMID,0);
/* Desatado de la zona de memoria compartida */
shmdt(tabla);
/* Cierre de la tubería utilizada */
close(idfifo);
}
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
PRACTICA 3 PRACTICA 3 PRACTICA 3 PRACTICA 3 PRACTICA 3
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
```

PROCESO LECTOR: Este proceso realiza la conversión a mayúsculas y mostrado en pantalla de los caracteres introducidos anteriormente por el proceso escritor en la zona de memoria compartida.

```
−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

#include <stdio.h>

#include <fcntl.h>

#include <sys/ipc.h>

#include <sys/types.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/stat.h>

#define SMUTEX 0 /* Semáforo de exclusión mutua */

#define SLLENO 1 /* Semáforo que indica las posiciones

ocupadas */#define SVACIO 2/*Semáforo que indica las posiciones vacias*/

#define TAM 512 /* Tamaño de la zona de memoria compartida */

main(int argc, char *argv[])

{

key_t llave; /* Llave de identificación de los mecanismos

IPC */ int shmid; /*Identificador de la zona de memoria

compartida */ int semid; /* Identificador del semáforo */ int idfifo; /* Identificador de la tubería */ char
*tabla; /* Tabla donde se almacenan los caracteres

escritos */

char dato; /* Variable que almacena el dato que hay

en memoria */ struct sembuf operacion[3]; /* Estructura de operaciones

sobre semáforos */ int i=0; /* Variable que indexa la tabla */

/* Abrimos la tubería en modo lectura y leemos la llave */

if ((idfifo=open("tuberia",O_RDONLY))==−1)

{
```

```
printf("Error al abrir la tuberia\n");
exit(−1);
}
if ((read(idfifo,&llave,sizeof(key_t)))==−1)
{
printf("\nError al leer de la tuberia\n");
exit(−1);
}
/* Creación de la zona de memoria compartida */
if ((shmid=shmget(llave,TAM,IPC_CREAT|0600))==−1)
{
printf("Error al crear la memoria compartida");
exit(−1);
}
/* Atarnos al segmento de memoria compartida */
tabla=shmat(shmid,0,0);
/* Creación del semáforo */
if ((semid=semget(llave,3,IPC_CREAT|0600))==−1)
{
printf("Error al crear los semaforos");
exit(−1);
}
do
{
/* P sobre los semáforos */
operacion[0].sem_num=SLLENO;
```

```
operacion[0].sem_op=−1;
operacion[0].sem_flg=0;
operacion[1].sem_num=SMUTEX;
operacion[1].sem_op=−1;
semop(semid,operacion,2);
/* Leemos el dato de la memoria */
dato=tabla[i];
/* V de los semáforos */
operacion[0].sem_num=SMUTEX;
operacion[0].sem_op=1;
operacion[1].sem_num=SVACIO;
operacion[1].sem_op=1;
semop(semid,operacion,2);
/* Al llegar al final reinicamos */
(i<TAM)?(i++):(i=0);
/* Conversión a mayúsculas y mostrado en pantalla */
if ((dato>='a')&&(dato<='z')) dato=dato−32;
if ((dato!='q')&&(dato!='Q')) printf("%c",dato);
}while ((dato!='q')&&(dato!='Q'));
printf("\n\n");
/* Borrado de los semáforos */
shmctl(shmid,IPC_RMID,0);
/* Desatado de la zona de memoria compartida */
shmdt(tabla);
/* Cierre de la tubería utilizada */
close(idfifo);
```

}

Cuestiones:

- En lugar de utilizar un semáforo para controlar el acceso al área compartida, describir un método usando el canal, para que el lector sepa en qué momento puede acceder a la memoria.

Podríamos, simplemente, mandar la llave por el canal una vez que el proceso escritor a finalizado de escribir, de este modo, el proceso lector comenzaría a ejecutarse al recibir la llave, pues espera a que llegue siempre.

- Por qué es necesario y suficiente que el lector conozca la llave para poder acceder al área de memoria compartida.

Porque una llave siempre tiene los mismos atributos para cualquier proceso que la maneje, esto es, misma memoria compartida, mismos semáforos, etc.

- ¿Funcionarían correctamente ambos procesos si el escritor en vez de pasar la llave, enviara a través del canal la dirección del segmento de memoria compartida? ¿Por qué?

No, porque para acceder a la memoria compartida es necesario hacerlos a través de la función `shmget' y esta necesita la llave, no basta simplemente conocer el identificador de la zona de memoria compartida. Además no compartirían los semáforos.

<u>PRÁCTICA 4</u>

```
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−− PRACTICA 4
PRACTICA 4 PRACTICA 4 PRACTICA 4 PRACTICA 4

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

PRODUCTOR−CONSUMIDOR:El productor recibir  datos n£mericos desde el teclado que ira depositando
en un buffer. El proceso consumidor los recogera del buffer y realizara la suma de dos en dos, almacenando el
resultado en el fichero 'suma.out'.

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

#include <pthread.h>

#include <stdio.h>

#include <fcntl.h>

#define LONG 16 /* Longitud del buffer */

typedef struct

{

int cantidad; /* Indica la cantidad de datos metidos */
```

```
int infor[LONG]; /* Contiene los numeros introducidos */
}t_buffer;
t_buffer buffer;
pthread_mutex_t semmutex, semproductor, semconsumidor;
pthread_mutexattr_t semmutexattr, semproductorattr,
semconsumidorattr;
void *productor(void *arg)
{
int dato; /* Numero leido */
int i=0; /* Indexa el buffer */
printf("\nIntroduce los numeros a sumar. Para finalizar
introduce −1\n");
do
{
scanf("%d",&dato); /* leo un numero */
pthread_mutex_lock(&semmutex); /* wait del semaforo */
if(buffer.cantidad<LONG) /* para no sobreescribir */
{
buffer.infor[i]=dato; /*guardo numero en buffer */
buffer.cantidad++;
i=(i+1)%LONG; /* Para que en 16 vuelva a 0 */
}
else dato=−1; /* nos salimos del proceso */
pthread_mutex_unlock(&semmutex);
}while(dato!=−1);
/* avisar al principal de que ha terminado el productor */
```

```
pthread_mutex_unlock(&semproductor);
}
void *consumidor(void *arg)
{
int num1,num2; /* Guardan los numeros a sumar */
int proces=1; /* Indica si sigo procesando los numeros */
int i=0; /* Indexa el buffer */
int fid; /* Identificador del fichero */
char cadena[256]; /* Almacena los caracteres de la suma */
if((fid=open("suma.out",O_CREAT|O_TRUNC|O_RDWR,0666))==-1)
{
perror("No se pudo abrir el fichero");
exit(-1);
}
do
{
pthread_mutex_lock(&semmutex);
switch(buffer.cantidad) /* Si hay datos para leer */
{
case 0:
break;
case 1:
if((buffer.infor[i])==-1) proces=0;
break;
default:
num1=buffer.infor[i];
```

```
i=(i+1)%LONG;
num2=buffer.infor[i];
i=(i+1)%LONG;
buffer.cantidad−=2;
if( num1==−1 || num2 ==−1) proces=0;
else proces=2;
}
pthread_mutex_unlock(&semmutex);
if(proces == 2 ) /* Tengo dos numeros a procesar */
{
proces=1; /* Sige en el bucle pero no guarda */
/*Almacenamos el resultado en cadena y lo guardamos*/
sprintf(cadena,"\t %−5d + %−5d = %d\n", num1, num2,
num1+num2);
write(fid,cadena,strlen(cadena));
}
}while(proces);
close(fid);
/* avisar al principal de que ha terminado el consumidor */
pthread_mutex_unlock(&semconsumidor);
}
main()
{
pthread_t th1,th2;
pthread_attr_t attr;
buffer.cantidad=0;
```

```
/* Creacion e inicializacion de los semaforos */
pthread_mutexattr_create(&semmutexattr);
pthread_mutexattr_create(&semproductorattr);
pthread_mutexattr_create(&semconsumidorattr);
if (pthread_mutex_init(&semmutex,semmutexattr)==−1 ||
pthread_mutex_init(&semproductor,semproductorattr)==−1 ||
pthread_mutex_init(&semconsumidor,semconsumidorattr)==−1)
{
perror("Error al inicializar los semaforos");
exit(−1);
}
/* Hacemos un wait de los semaforos de productor y consumidor */
pthread_mutex_lock(&semproductor);
pthread_mutex_lock(&semconsumidor);
/* Creacion de los atributos de los hilos */
if (pthread_attr_create(&attr) == −1)
{
perror("Error al crear los attributos de los hilos");
exit(−1);
}
/* Creacion de los hilos */
if (pthread_create(&th1,attr,productor,NULL) == −1 ||
pthread_create(&th2,attr,consumidor,NULL) == −1)
{
perror("Error al crear los hilos");
exit(−1);
```

```
}

/* Hacemos un wait de los semaforos de productor y consumidor para esperar a que terminen los dos procesos
*/

pthread_mutex_lock(&semproductor);

pthread_mutex_lock(&semconsumidor);

printf("\nResultado:\n");

system("cat suma.out");

exit(0);

}
```

Cuestiones:

- Compile el programa de ejemplo (hilo.c) y compruebe los resultado de su ejecución. ¿Cuantos hilos llegan a estar activos simultáneamente?

Tres, el hilo principal y dos secundarios.

- En el ejemplo sincro.c, con que valor se inicializan los semáforos? Compruebe si los semáforos pueden tomar valores positivos superiores a la unidad. Muestre un programa de ejemplo (comprueba.c) que verifique si lo anterior es o no cierto.

Se inicializan siempre a uno.

```
/*−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−

PRACTICA 4 PRACTICA 4 PRACTICA 4 PRACTICA 4 PRACTICA 4

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

PROCESO COMPROBACION: Este proceso comprueba los posibles valores que pueden tomar los
semáforos.

−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−*/

#include <pthread.h>

#include <stdio.h>

pthread_mutex_t semaforo1,semaforo2;

pthread_mutexattr_t semaforoattr1,semaforoattr2;

/* En este caso es para ver el caso de que tomen valores negativos */

void *caso1 (void *arg)
```

```
{
printf("Estamos en el primer caso:\n");
printf("Inicialmente el semaforo1 = 1 \n");
pthread_mutex_lock(&semaforo1);
printf("wait del semaforo1\n");
printf("el semaforo1 pasa a valer 0 \n\n");
pthread_mutex_lock(&semaforo1);
printf("wait semaforo 1\n");
printf("El semaforo1 vale −1 Y ESTO NO FUNCIONA \n");
}
/* En este caso es para ver el caso de que tomen valores positivos */
void *caso2 (void *arg)
{
printf("Estamos en el segundo caso:\n");
printf("Inicialmente el semaforo2 = 1 \n");
pthread_mutex_unlock(&semaforo2);
printf("signal del semaforo2 => semaforo2 = 2\n");
pthread_mutex_unlock(&semaforo2);
printf("signal del semaforo2 => semaforo2 = 3\n");
pthread_mutex_lock(&semaforo2);
printf("wait semaforo 2 => semaforo2 = 2 \n");
printf("si no hay bloqueo => TOMAN VALORES SUPERIORES A LA
UNIDAD \n\n");
}
main()
{
```

```
pthread_t th1,th2;
pthread_attr_t attr;
pthread_mutexattr_create(&semaforoattr1);
pthread_mutexattr_create(&semaforoattr2);
if( pthread_mutex_init(&semaforo1,semaforoattr1) == −1 )
{
perror("Error en la creacion del semaforo 1");
exit(1);
}
if( pthread_mutex_init(&semaforo2,semaforoattr2) == −1 )
{
perror("Error en la creacion del semaforo 2");
exit(1);
}
if( pthread_attr_create(&attr) == −1 )
{
perror("Error en la creacion de los atributos de los
hilos");
exit(1);
}
if( pthread_create(&th1,attr,caso1,NULL) == −1 )
{
perror("Error en la creacion del hilo 1");
exit(1);
}
if( pthread_create(&th2,attr,caso2,NULL) == −1 )
```

```
{

perror("Error en la creacion del hilo 2");

exit(1);

}

printf("El hilo principal se duerme durante 30seg\n");

sleep(30);

printf("\n Salida del hilo principal\n");

exit(0);

}
```

- En el programa del productor consumidor, qué ocurre si el consumidor va más lento que el productor. Introduzca un retardo en el código del consumidor para comprobar qué ocurre.

Lo que ocurre es que los datos que quiera introducir el consumidor en el buffer no podrán ser metidos debido a que está lleno.

- ¿Qué ocurre si el productor es más lento?

Si el productor es el más lento, no ocurre nada, dado que no empieza a procesar hasta que no produzca los datos el productor.