Trabajo de Diseño y Administración de Sistemas Operativos. Comunicando Procesos.

Manuel Rodríguez Sánchez

mrodrigue212@alumno.uned.es

Contenido

1 Introducción

Desarrollo de un proyecto en C, donde se muestra un ejemplo de la comunicación y sincronización entre procesos mediante el envío de un mensaje, y como éste se va moviendo por cada proceso mediante tuberías, colas de mensajes y memoria compartida sincronizada con un semáforo, de acuerdo al siguiente esquema

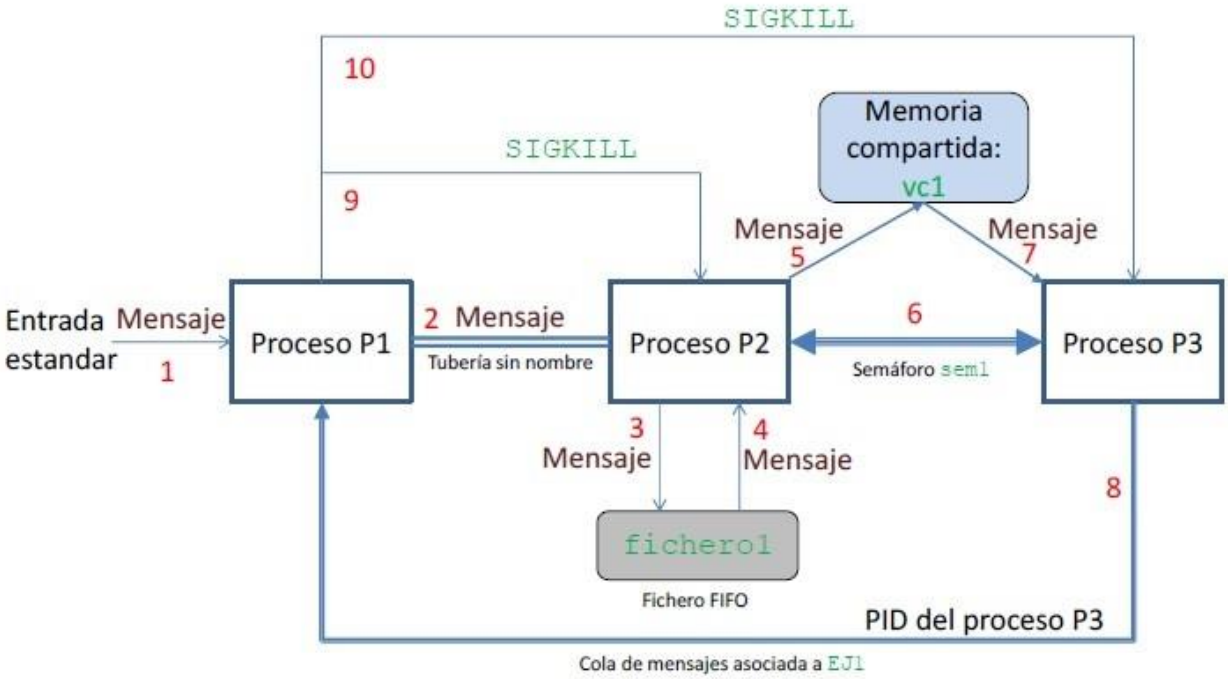

Figura1: Esquema de comunicación y sincronización entre los procesos.

2 Implementación

Se han implementado cuatro ficheros fuentes, tres de ellos en lenguaje C y un cuarto fichero en script BASH.

El fichero Ejercicio2.sh es el que se ha implementado en script BASH, y éste compila los tres ficheros fuentes en C, generando los correspondientes ejecutables Ej1, Ej2 y Ej3, modificando posteriormente los permisos de éstos, y ejecutando el Ej1 que pondrá en funcionamiento todo el esquema indicado en la figura 1 del enunciado del ejercicio.

Código fuente del fichero Ejercicio2.sh:

```
clear #limpiamos pantalla
cd Trabajo2
rm -f fichero1 >> /dev/null #por si acaso existe, eliminarlo
# Compilamos ficheros fuente
gcc -o Ej1 fuente1.c
gcc -o Ej2 fuente2.c
gcc -o Ej3 fuente3.c
#Modificamos los permisos de los nuevos ficheros
chmod 777 Ej1
chmod 777 Ej2
chmod 777 Ej3
# Para que nos muestre el tiempo de ejecución cuando finalice
time ./Ej1
#Eliminamos los ejecutables
rm Ej1
rm Ej2
rm Ej3
```

Los ficheros fuentes en C: ***fuente1.c, fuente2.c y fuente3.c,*** contienen todas las instrucciones necesarias para crear los procesos, hacer las llamadas al sistema, crear los mecanismos de comunicación IPC del system V de comunicación y sincronización entre procesos.

Fichero fuente1.c

El código fuente de este fichero, cuyo ejecutable es Ej1, crea dos procesos, donde cada proceso realiza una serie de acciones. A continuación se especifica la acción y en que línea del código fuente en lenguaje C, se produce ésta:

1º Al ejecutarse Ej1, se crea un proceso (proceso 1) donde se crea una tubería sin nombre con la llamada pipe (tubería) de la línea 58. Se hace la llamada al sistema para la creación de un nuevo proceso (proceso 2) con fork () (línea 59). En el proceso 1 se pide que se introduzca un mensaje (línea 65 y 66). El tamaño máximo del mensaje viene establecido por la definición de la constante SIZE, con un tamaño de 512 bytes (línea 17).

2º Mediante la tubería sin nombre creada, se transmite el mensaje del proceso 1 al proceso 2. Se escribe en la línea 70 (proceso 1), y se recoge o lee en la línea 114 (proceso 2).

3º El proceso 1 crea una cola de mensajes, con su correspondiente llave (líneas 74 a 84), quedando a la espera del PID del proceso 3, que se lo enviará éste a través de esta cola de mensajes.

4º El proceso 1, mandará señales para la eliminación de los procesos 2 y 3 (líneas 88 a 90), eliminará el fichero FIFO creado en proceso 2 (líneas 93 y 94) y eliminará la cola de mensajes (líneas 97 y 98).

5º Por último, el proceso 1, calculará y mostrará los cálculos de los tiempos de uso de la CPU (líneas 102 a 105).

6º El proceso 2, lee el mensaje de la tubería sin nombre (línea 114) y lo muestra por pantalla.

7º El proceso 2, crea un fichero FIFO (fichero1) y guarda el mensaje en este fichero (línea 124 a 133).

8º El proceso 2, ejecuta el Ej2 que corresponde al ejecutable generado por el *fuente2.c*

Código fuente:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <libgen.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/times.h>
#define SIZE 512
#define ESCRITURA 1
#define LECTURA 0

//Procedimiento con información del error, su origen, proceso y causa
void error(char *origenError, int numeroProceso){
        printf("%s. Proceso: %i. Información del sistema: %s\n", origenError, numeroProceso, strerror(errno));
        exit(-1);
}
```

```
int main( int argc, char **argv )
{
	//Declaraciones
	int tuberia[2], tamanoMensaje, proceso, longitud, contador, bytesMensaje;
	char mensaje[SIZE];
	int fichero;
	int msqid; //indicador de la cola de mensajes asociado a la llave

	/********** INSTRUCCIONES PARA LAS ESTADÍSTICAS DE CPU ********************/

	//Estructuras que almacenarán información estadística de los tiempos de ejecución empleados por el proceso.
	struct tms infTemp1,infTemp2;
	//Variables para almacenar los tiempos de inicio y fin.
	clock_t tiempo1, tiempo2;
	//Rellenamos la estructura infTemp1 con la informacion estadística de los tiempos de ejecución iniciales
	tiempo1=times(&infTemp1);

	/*****************************************************************************/

	// estructura para la cola de mensajes
	struct
	{
		long idMensaje;
		int pid_3;//Contiene el PID del proceso 3
	}colaDeMensajes;

	key_t llave;
	pid_t pid2, pid3;


	pipe(tuberia);//creamos la tuberia
	if((pid2=fork())==-1) error("Imposible crear proceso 2\n",1);

	if  (pid2 != 0) // Estamos en el proceso 1
	{ // padre
		proceso=1;
		close( tuberia[LECTURA] ); /* cerramos el lado de lectura del pipe */
		printf("Introduce el mensaje: ");
		gets(mensaje);
```

```
strcat(mensaje,"."); //Agregamos un punto al final del mensaje

//Escribimos en la tubería sin nombre
if(write(tuberia[ESCRITURA], mensaje, (strlen(mensaje)))<0) error("No se puede escribir en la tuberia sin nombre\n",proceso);
printf("El proceso %d (PID=%d, Ej1) transmite mensaje '%s' al proceso 2 por una tubería sin nombre.\n", proceso, getpid(),mensaje);

//Creación de cola de mensajes
llave=ftok("Ej1",'A');
if ((msqid=msgget(llave,0666|IPC_CREAT))<0) error("No se puede crear la cola de mensajes\n",proceso);

//Estas líneas esperan el mensaje que ha de llegar por la cola desde el proceso 3
bytesMensaje=0;
while (bytesMensaje<=0){
    bytesMensaje=msgrcv(msqid,(struct msgbuf *)&colaDeMensajes,sizeof(colaDeMensajes),1,0);
    if (bytesMensaje<0) error("Ha habido problemas con la recepción de mensajes\n",proceso);

}
pid3=colaDeMensajes.pid_3;
printf("El proceso %d (PID=%d, Ej1) ha recibido por cola de mensajes el PID %i del proceso 3.\n", proceso, getpid(), pid3);

//Señales para matar procesos y finalizar
printf("El proceso %d (PID=%d, Ej1) Envía señales para matar el proceso 2(PID %d) y el proceso 3(PID %d)\n", proceso, getpid(), pid2, pid3);
if (kill(pid3,SIGKILL)<0) error("No se puede eliminar proceso 3.\n", proceso);
if (kill(pid2,SIGKILL)<0) error("No se puede eliminar proceso 2.\n", proceso);

//Eliminación del fichero FIFO
printf("El proceso %d (PID=%d, Ej1) elimina 'fichero1'.\n",proceso, getpid());
if (unlink("fichero1")<0) error("No se puede eliminar fichero1",proceso);

//Eliminación de la cola de mensajes
msgctl(msqid,IPC_RMID,0);
printf("El proceso %d (PID=%d, Ej1) elimina la cola de mensajes.\n",proceso, getpid());

/********** Cálculos para las estadísticas de uso de CPU *********/
```

```
		tiempo2=times(&infTemp2); // Rellenamos la segunda estructura para el cálculo de inicio y fin.

		printf("\nTiempo total de uso de la CPU: %g\n", (float)(tiempo2-tiempo1)/CLOCKS_PER_SEC);

		printf("Tiempo de uso de la CPU en modo usuario: %g\n",(float)(infTemp2.tms_utime-
infTemp1.tms_utime)/CLOCKS_PER_SEC);
		printf("Tiempo de uso de la CPU en modo núcleo: %g\n",(float)(infTemp2.tms_stime-
infTemp1.tms_stime)/CLOCKS_PER_SEC);


	}
	else // Estamos en el proceso 2
	{ // hijo
		proceso=2;
		close( tuberia[ESCRITURA] ); /* cerramos el lado de escritura del pipe */

		//leemos el mensaje de la tuberia sin nombre, mientras no haya datos, no lee nada
		tamanoMensaje=read( tuberia[LECTURA], mensaje, SIZE );

		printf("El proceso %d (PID=%d, Ej1) recibe mensaje:'", proceso, getpid());
		for (contador=0;contador<tamanoMensaje;contador++) printf("%c", mensaje[contador]);
		printf("', del proceso 1 por una tubería sin nombre.\n");

		//Creando fichero de tuberia con nombre "fichero1"
		if(mknod("fichero1", S_IFIFO|0666,0)<0) error("No se puede crear el fichero\n",proceso);

		//apertura del fichero
		fichero=open("fichero1",O_RDWR|O_NONBLOCK);
		if (fichero<0) error("No se ha podido abrir el fichero1\n",proceso);

		//Se escribe mensaje en el fichero
		printf("El proceso %d (PID=%d, Ej1) coloca el mensaje:'", proceso, getpid());
		//Para sacar el mensaje sin basura
		for (contador=0;contador<tamanoMensaje;contador++) printf("%c", mensaje[contador]);
		printf("', en un fichero FIFO llamado 'fichero1'.\n");
		if (longitud=write(fichero,mensaje,strlen(mensaje))<0) error("Error al escribir en
fichero1\n",proceso);
		execv("./Ej2",NULL);
	/*************************************************************************************/
	}
	return(0);
}
```

Fichero fuente2.c

Dentro de este fichero, cuyo ejecutable es Ej2, se ejecutan parte de las instrucciones correspondientes al proceso 2 creado en el *fuente1.c*. En este mismo fichero, se creará un tercer proceso (proceso 3, línea 77) que hará la llamada al Ej3. Las acciones realizadas en este fichero son:

1º El proceso 2 hace la apertura y lectura del fichero FIFO (fichero1) (líneas 48 a 53).

2º El proceso 2 crea una llave (líneas 56 a 60) para el semáforo y el segmento de memoria compartida que se crearán posteriormente.

3º Proceso 2 crea espacio de memoria compartida (líneas 63 y 64) creando el identificador de la zona de memoria y la dirección a la que está unido el segmento de memoria compartida.

4º Proceso 2 crea el semáforo (línea 67)

5º El proceso 2 inicializa el semáforo (líneas 70 y 71) y lo pone en 'rojo' de forma que bloquea el acceso del proceso 3, es decir, como el proceso 2 ha de escribir el mensaje en memoria, para evitar un acceso de lectura de mensaje del proceso 3, el semáforo lo bloqueamos o ponemos en rojo. Usamos la variable opSem, que apunta a la estructura con el valor de la variable op_sem=1 (rojo) (linea 72). Con el semáforo sincronizamos la escritura del proceso 2 y la lectura del futuro proceso 3.

6º El proceso 2 hace la llamada al sistema fork() para la creación de un nuevo proceso (proceso 3, línea 75), a partir de este punto, hay una bifurcación del proceso 2 y 3.

7º El proceso 2 (a partir de línea 82) provoca que el sistema duerma 1 segundo (línea 84) y posteriormente coloca el mensaje en el espacio de memoria compartida (líneas 87 y 88), dejándolo disponible para que lo lea el proceso 3. De esta forma sincronizamos los procesos para que no se pisen.

8º El proceso 2 pone el semáforo en 'verde' restándole 1 a la variable de la estructura (opSem.sem_op=-1), se hace la llamada a la operación (líneas 91 y 92).

9º El proceso 2 se pausa esperando ser matado por el proceso 1(línea 96).

10º El proceso 3 ejecuta Ej3 (línea 80).

Código fuente:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <libgen.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/msg.h>
#include <sys/ipc.h>
```

```
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/times.h>
#define SIZE 512

//Procedimiento con información del error, su origen, proceso donde se genera y causa.
void error(char *origenError, int numeroProceso){
	printf("%s. Proceso: %i. Información del sistema: %s\n", origenError, numeroProceso, strerror(errno));
	exit(-1);
}

int main(int argc, char *argv[])
{
	//Declaraciones
	char mensaje[SIZE];
	int fichero, sem1, *vc1, tamanoMensaje, proceso, shmId, contador, contador2;

	/*shmId: identificador del espacio de memoria compartida
	sem1: identificador del semáforo*/

	key_t llaveSM; //Llave para identificacion de Semaforo y Memoria compartida
	pid_t pid3; //Almacena el número de pid del proceso 3

	union semun{
		int val;
		struct semin_ds *buf;
		ushort* array;
	}arg;

	struct sembuf opSem = {0,1,0};//Estructura para operaciones de bloqueo/desbloqueo en el semáforo. Inicializamos a 1.

	proceso=2;

	//Apertura fichero FIFO.
	fichero=open("fichero1", O_RDWR|O_NONBLOCK);
	if(fichero<0) error("Error en la apertura de fichero1.\n",proceso);

	//Lectura del mensaje almacenado en fichero FIFO 'fichero1'
	if ((tamanoMensaje=read(fichero,mensaje,SIZE))<0) error("Error al leer el fichero.\n",proceso);
```

```
printf("El proceso %d (PID=%d, Ej2) lee el mensaje: %s, del fichero FIFO 'fichero1'.\n",proceso, getpid(), mensaje);

// Creación de llave para semáforo y memoria compartida
llaveSM=ftok("fichero1",'B');
if (llaveSM==(key_t)-1)
{
        error("No se ha podido crear la llave.\n",proceso);
}

//Creación de espacio de memoria compartida
if ((shmId=shmget(llaveSM,SIZE,0666|IPC_CREAT))<0) error("Error al generar el identificador de la zona de memoria compartida asociada a la llave.\n",proceso);
if((vc1=(int *)shmat(shmId,0,0))<0) error("No se ha podido almacenar la dirección a la que está unido el segmento de memoria compartida.\n", proceso);

// Creación de semáforo
if ((sem1=semget(llaveSM,1,0666|IPC_CREAT))<0) error("No se ha podido crear el conjunto de semáforos.\n",proceso);

//Iniciacilización del semáforo
arg.val=0;
if((semctl(sem1,0,SETVAL,arg))==-1) error("No se puede inicializar el semáforo.\n", proceso);
semop(sem1,&opSem,1);//Bloqueado acceso a Proceso 3

//Bifurcación nuevo proceso P3
if ((pid3=fork())==-1) error("No se ha podido crear el proceso 3\n", proceso);

if (pid3==0) //Estamos en el proceso 3
{
        proceso=3;
        execv("./Ej3",NULL);
}
else //Proceso 2
{
        sleep(1);
        printf("El proceso 2 (PID=%d, Ej2) coloca el mensaje: '%s', en un espacio de memoria compartido\n", getpid(), mensaje);
        //Metemos el mensaje en el espacio de memoria compartida
        vc1[0]=strlen(mensaje);
        for (contador=0;contador<strlen(mensaje);contador++) vc1[contador+1]=(int) mensaje[contador];
```

```
        //Como ya hemos escrito el mensaje en memoria, el semáforo se abre para permitir acceso a Proceso 3
                opSem.sem_op=-1;
                semop(sem1,&opSem,1);

                //Proceso 2 esperando ser asesinado
                printf("El proceso 2 (PID=%d, Ej2) queda a la espera de ser matado.\n",getpid());
                pause();
        }
        return(0);
}
```

Fichero fuente3.c

Dentro de este fichero cuyo ejecutable es Ej3, se ejecutan instrucciones correspondientes al proceso 3, el cual fue creado en el *fuente2.c*. Las acciones realizadas en este fichero son:

1º El proceso 3 coge el semáforo 'sem1', generado en el proceso 2, y que está asociado al segmento de memoria (línea 41).

2º El proceso 3 crea y abre el espacio de memoria compartida, que es donde está el mensaje, mediante la variable memoriaId y vc1 (variable que almacena la dirección a la que está unida el segmento de memoria creado, es decir memoriaId) (Líneas 44 y 45).

3º El proceso 3 espera a que el proceso 2 ponga el semáforo en verde, para que el proceso 3 pueda leer (línea 48).

4º Una vez cambia el semáforo en el proceso 2, el proceso 3 lee el mensaje y lo muestra por pantalla (líneas 51 a 53).

5º Proceso 3 desasocia el puntero del espacio de memoria compartida (línea 56), y elimina ésta (línea 58). Se elimina también el semáforo en línea 62.

6º El proceso 3 manda mediante la cola de mensajes, el PID al proceso 1 (líneas 66 a 69) y se envía mediante la llave y la estructura (líneas 73 a 82).

7º El proceso 3 queda pausado hasta ser matado por el proceso 1 (línea 87).

Código fuente:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include <signal.h>
#include <errno.h>
#include <libgen.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <sys/msg.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/times.h>
#define SIZE 512

//Procedimiento con información del error, su origen, proceso y causa
void error(char *origenError, int numeroProceso){
	printf("%s. Proceso: %i. Información del sistema: %s\n", origenError, numeroProceso, strerror(errno));
	exit(-1);
}

int main(int argc, char *argv[])
{
	int llaveSM, sem1, memoriaId, *vc1, contador, pid3, proceso, longitud;
	char mensaje[SIZE];

	struct // estructura para la cola de mensajes
	{
		long idMensaje;
		int pid_3;
	}colaDeMensajes;

	proceso=3;

	llaveSM=ftok("fichero1",'B');

	//Asignación de Semáforo
	if((sem1=semget(llaveSM,1,0666))<0) error("Error en la asignación de semáforo",proceso);

	//Abrimos espacio de memoria compartida
	if((memoriaId=shmget(llaveSM,SIZE,0666))<0) error("Error en la creación de la zona de memoria compartida\n",proceso);
	if((vc1=(int *)shmat(memoriaId,0,0))<0) error("Error al almacenar la dirección a la que está unida el segmento de memoria compartida\n",proceso);

	//Esperamos a que el proceso 2, cambie el semáforo para leer de la memoria.
	while(semctl(sem1,0,GETVAL,0)>0);

	//Ha cambiado el semáforo y da acceso al proceso 3, leyendo éste el mensaje
```

```
	longitud=vc1[0];
	for (contador=0;contador<longitud;contador++) mensaje[contador]=(char) vc1[contador+1];
	printf("El proceso %d (PID=%d, Ej3) lee de la memoria compartida el mensaje: '%s'.\n", proceso, getpid(), mensaje);

	//Desasociamos el puntero del espacio de memoria compartida
	shmdt((char *)vc1);
	//Eliminamos la zona de memoria compartida
	shmctl(memoriaId,IPC_RMID,0);
	printf("El proceso %d (PID=%d, Ej3) libera la zona de memoria compartida.\n",proceso, getpid());

	//Eliminamos el semáforo
	semctl(sem1,0,IPC_RMID);
	printf("El proceso %d (PID=%d, Ej3) elimina semáforo.\n",proceso, getpid());

	//Enviamos PID al proceso 1 por la cola de mensajes
	pid3=getpid();
	colaDeMensajes.idMensaje=1;
	colaDeMensajes.pid_3=pid3;
	printf("El proceso %d (PID=%d, Ej3) envía al proceso 1 su PID por cola de mensajes.\n", proceso, getpid());


	//Abrimos la cola para el envío
	int llaveCola=ftok("Ej1",'A');
	int msqid;
	if (llaveCola == (key_t)-1)
	{
		error("Error al obtener llave para cola mensajes\n",proceso);
	}
	//Obtenemos el identificador de la cola de mensajes.
	if((msqid=msgget(llaveCola,0666|IPC_CREAT))<0) error("Error al obtener identificador para cola mensajes\n",proceso);
	//Enviamos el mensaje
	if(msgsnd(msqid,(struct msgbuf *)&colaDeMensajes,sizeof(colaDeMensajes),0)<0) error("No se puede enviar PID por cola de mensajes\n",proceso);
	//Pausamos el sistema a la espera de ser asesinado.
	printf("El proceso %d (PID=%d, Ej3) queda a la espera de ser matado.", proceso, getpid());
	pause();
	return (0);
}
```

3 Ejecución de ejemplo

A continuación, podemos observar una ejecución de ejemplo:

Introduce el mensaje: Esto es un mensaje

El proceso 1 (PID=2882, Ej1) transmite mensaje 'Esto es un mensaje.' al proceso 2 por una tubería sin nombre.

El proceso 2 (PID=2883, Ej1) recibe mensaje:'Esto es un mensaje.', del proceso 1 por una tubería sin nombre.

El proceso 2 (PID=2883, Ej1) coloca el mensaje:'Esto es un mensaje.', en un fichero FIFO llamado 'fichero1'.

El proceso 2 (PID=2883, Ej2) lee el mensaje: Esto es un mensaje., del fichero FIFO 'fichero1'.

El proceso 2 (PID=2883, Ej2) coloca el mensaje: 'Esto es un mensaje.', en un espacio de memoria compartido

El proceso 2 (PID=2883, Ej2) queda a la espera de ser matado.

El proceso 3 (PID=2886, Ej3) lee de la memoria compartida el mensaje: 'Esto es un mensaje.'.

El proceso 3 (PID=2886, Ej3) libera la zona de memoria compartida.

El proceso 3 (PID=2886, Ej3) elimina semáforo.

El proceso 3 (PID=2886, Ej3) envía al proceso 1 su PID por cola de mensajes.

El proceso 1 (PID=2882, Ej1) ha recibido por cola de mensajes el PID 2886 del proceso 3.

El proceso 1 (PID=2882, Ej1) Envía señales para matar el proceso 2(PID 2883) y el proceso 3(PID 2886)

El proceso 1 (PID=2882, Ej1) elimina 'fichero1'.

El proceso 1 (PID=2882, Ej1) elimina la cola de mensajes.

Tiempo total de uso de la CPU: 0.000853

Tiempo de uso de la CPU en modo usuario: 0

Tiempo de uso de la CPU en modo núcleo: 0

real 0m8.531s

user 0m0.000s

sys 0m0.000s

Bibliografía.

Fundamentos del sistema operativo Unix. José Manuel Díaz Martínez, Rocío Muñoz Mansilla y Dictino Chaos García

Manual de Borland C/C++. Chris H. Papas, William H. Murray. Osborne McGraw -Hill.

El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie. Pretince Hall Hispanoamérica S.A.

Sistemas Operativos, teoría y problemas. J. Aranda, M.A. Canto, J.M. De la Cruz, S. Dormido,C. Mañoso. UNED. Editorial Sanz y Torres.

Unix y Linux, guía práctica (2ª Ed). Sebastián Sánchez. Ed. RA-MA.

Webs

http://www.programacion.com.py/escritorio/c/pipes-en-c-linux

http://www.lsi.us.es/cursos/seminario-1.html

http://www.chuidiang.com/clinux/ipcs/colas.php

http://www.chuidiang.com/clinux/ipcs/semaforo.php

http://www.chuidiang.com/clinux/ipcs/mem_comp.php

http://totaki.com/poesiabinaria/2009/07/tuberias-con-nombre-para-comunicacion-entre-procesos/

http://stackoverflow.com/