UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE INFORMÁTICA Y ANÁLISIS NUMÉRICO

ASIGNATURA ***SISTEMAS OPERATIVOS***

2º DE GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 4

Memoria compartida

Profesorado: Juan Carlos Fernández Caballero

Enrique García Salcines

Índice de contenido

1 Objetivo de la práctica

La presente práctica persigue familiarizar al alumnado con la comunicación inter-procesos utilizando el método de memoria compartida. En un primera parte se explicará brevemente teoría sobre memoria compartida, siendo en la segunda parte de la práctica cuando, mediante programación en C, se practicarán los conceptos sobre esta temática, utilizando para ello las rutinas de interfaz del sistema que proporcionan a los programadores la librería *shm* basada en el estándar POSIX.

2 Recomendaciones

El lector debe completar las nociones dadas en las siguientes secciones con consultas bibliográficas, tanto en la Web como en la biblioteca de la Universidad, ya que unos de los objetivos de las prácticas es potenciar su capacidad autodidacta y su capacidad de análisis de un problema. Es recomendable que, aparte de los ejercicios prácticos que se proponen, pruebe y modifique otros que se encuentren en la Web (se dispone de una gran cantidad de problemas resueltos en C sobre esta temática), ya que al final de curso deberá acometer un examen práctico en ordenador como parte de la evaluación de la asignatura.

No olvide que debe consultar siempre que lo necesite el estándar POSIX http://pubs.opengroup.org/onlinepubs/9699919799/ y la librería GNU C (*glibc*) en http://www.gnu.org/software/libc/libc.html.

Los conceptos teóricos que se exponen a continuación se aplican tanto a procesos como a hilos, pero la parte práctica de este guión va dirigida a la resolución de problemas de concurrencia mediante hilos. Para obtener soluciones usando procesos el lector debería utilizar las rutinas de interfaz para memoria compartida entre procesos que proporcione el sistema que esté utilizando. POSIX (*POSIX Shared Memory)*[1] proporciona también un estándar para ello que implementan los sistemas basados en UNIX.

3 Conceptos teóricos

3.1 Conceptos generales de memoria compartida.

La forma más eficaz que tienen los procesos para comunicarse consiste en compartir una zona de memoria, tal que para enviar datos de un proceso a otro, sólo se han de escribir en dicha memoria y automáticamente estos datos estarán disponibles para cualquier otro proceso. La utilización de este espacio de memoria común evita la duplicación de datos y el lento trasvase de información entre los procesos.

La memoria convencional que puede direccionar un proceso a través de su espacio de direcciones virtuales es local a ese proceso y cualquier intento de direccionar esa memoria desde otro proceso va a provocar una violación de segmento. Es decir, cuando se crea uno o más procesos mediante la llamada *fork( )*, se realiza una duplicación de todas las variables usadas, de forma que cualquier modificación de sus valores pasa inadvertida a los demás procesos, puesto que aunque el nombre es el mismo, su ubicación en memoria no lo es. Esto es debido a que con cada nuevo proceso se reserva una zona de memoria inaccesible a los demás. Las direcciones de las variables

1 http://en.wikipedia.org/wiki/Shared_memory

de esta zona son virtuales, y es el módulo de gestión de la memoria el que se encarga de traducirlas a direcciones físicas.

Para solucionar este problema, Linux ofrece la posibilidad de crear zonas de memoria con la característica de poder ser direccionadas por varios procesos simultáneamente. A estas zonas de memoria se les denominada segmentos de memoria compartida. Un proceso, antes de usar un segmento de memoria compartida, tiene que atribuirle una dirección virtual en su espacio de direcciones (mapearlo, sección 3.4), con el fin de que dicho proceso pueda acceder a él por medio de esta dirección. Esta operación se conoce como conexión, unión o enganche de un segmento con el proceso.

El kernel no gestiona de forma automática los segmentos de memoria compartida, por lo tanto, es necesario asociar a cada segmento o grupo de segmentos un conjunto de mecanismos de sincronización, de forma que se evite los típicos choques entre los procesos, es decir, que un proceso este modificando la información contenida en el segmento mientras que otro lee sin que se hayan actualizado todos los datos.

Claves IPC

Antes de comenzar a estudiar las llamadas al sistema para la utilización de memoria compartida es necesario explicar que son las claves IPC (InterProcess Communication). Cada objeto IPC tiene un identificador único, que se usa dentro del núcleo para identificar de forma única un objeto IPC. Una zona de memoria compartida es un objeto IPC, ya que es un objeto que sirve para la comunicación entre procesos (otros objetos IPC son los mensajes o los semáforos).

Para obtener el identificador único correspondiente a un objeto IPC se debe utilizar una *clave*. Esta clave debe ser conocida por los procesos que utilizan el objeto. La clave puede ser estática, incluyendo su código en la propia aplicación. Pero en este caso hay que tener cuidado ya que la clave podría estar en uso. Otra forma es generar la clave utilizando la función `ftok` cuyo prototipo es el siguiente:

```
key_t ftok(char *nombre, char proj);
```

`ftok` devuelve una clave basada en `nombre` y `proj` que puede ser utilizada para la obtención de un identificador único de objeto IPC. El argumento `nombre` debe ser el nombre de camino de un fichero existente. Esta función devolverá la misma clave para todos los caminos que nombren el mismo fichero, cuando se llama con el mismo valor para `proj`. Devolverá valores diferentes para la clave cuando es llamada con caminos que nombren a distintos ficheros o con valores diferentes para el parámetro `proj`.

Ejemplo:

```
key_t clave;
clave=ftok(“.”,’S’);
```

3.2 Petición de un segmento de memoria compartida

La función *shmget*[2] permite crear una zona de memoria compartida o habilitar el acceso a una ya creada. Su declaración es la siguiente:

```
#include <sys/shm.h>
int shmget (key, size, flag);

key_t key;      /* clave del segmento de memoria compartida */
int size;       /* tamaño del segmento de memoria compartida */
int flag;       /* opción para la creación */
```

El parámetro *clave* es la clave IPC o llave de acceso tal y como se explicó en el apartado 3.1.

El segundo parámetro, *size*, indica el espacio en bytes de la zona de memoria compartida que se desea crear. Este espacio es fijo durante su utilización.

El último parámetro *flag*, es una máscara de bits que definirá los permisos de acceso a la zona de memoria y el modo de adquirir el identificador de la misma. Puede tomar los siguientes valores:

- `IPC_CREAT`: crea un segmento si no existe ya en el núcleo.
- `IPC_EXCL`: al usarlo con `IPC_CREAT`, falla si el segmento ya existe (IPC_CREAT| IPC_EXCL)

Devuelve el identificador del segmento de memoria compartida si éxito o –1 en caso de error.

El identificador del segmento que devuelve esta función es heredado por todos los procesos descendientes del que llama a esta función.

El siguiente trozo de código muestra cómo se crea una zona de memoria de 4096 bytes, donde sólo el propietario va a tener permiso de lectura y escritura:

```
#define LLAVE (key_t)  234 /* clave de acceso */
int shmid; /* identificador del nuevo segmento de memoria compartida */
if((shmid=shmget(LLAVE, 4096, IPC_CREAT | 0600)) == -1)
{
   /* Error al crear o habilitar el segmento de memoria compartida. Tratamiento del error. */
}
```

3.3 Control de un segmento de memoria compartida

La función *shmctl*[3] proporciona información administrativa y de control sobre el segmento de memoria compartida que se especifique. Su declaración es la siguiente:

2 http://pubs.opengroup.org/onlinepubs/009695399/functions/shmget.html
3 http://pubs.opengroup.org/onlinepubs/009695399/functions/shmctl.html

```
#include <sys/shm.h>
int shmctl (shmid, cmd, *buf);

int shmid;                  /* identificador del segmento */
int cmd;                    /* operación a efectuar */
struct shnud_ds *buf;       /* información asociada al segmento de memoria compartida  */
```

Donde `shmid` es el identificador de la zona de memoria compartida, `cmd` es la operación que se quiere realizar y `buf` es una estructura donde se guarda la información asociada al segmento de memoria compartida en caso de que la operación sea `IPC_STAT` o `IPC_SET`. Las operaciones que se pueden realizar son:

IPC_STAT: guarda la información asociada al segmento de memoria compartida en la estructura apuntada por buf. Esta información es por ejemplo el tamaño del segmento, el identificador del proceso que lo ha creado, los permisos, etc.

IPC_SET: Establece los permisos del segmento de memoria a los de la estructura `buf`.

IPC_RMID: Marca el segmento para borrado. No se borra hasta que no haya ningún proceso que esté asociado a él.

La llamada a esta función devuelve el valor 0 si se ejecuta satisfactoriamente, y –1 si se ha producido algún fallo.

El siguiente trozo de código muestra como se borra un segmento de memoria compartida previamente creado:

```
int shmid; /* identificador del segmento de memoria compartida */
....
if(shmctl (shmid, IPC_RMID,NULL) == -1)
{
    /* Error al eliminar el segmento de memoria compartida. Tratamiento del error. */
}
```

3.4 Mapeado del segmento de memoria compartida

Un proceso, antes de usar un segmento de memoria compartida, tiene que atribuirle una dirección virtual en su espacio de direcciones (mapearlo), con el fin de que dicho proceso pueda acceder a él por medio de esta dirección. Esta operación se conoce como conexión, unión o enganche de un segmento con el proceso. Una vez que el proceso deja de usar el segmento, debe de realizar la operación inversa (desconexión, desunión o desenganche), dejando de estar accesible el segmento para el proceso.

La función *shmat*[4] realiza la conexión del segmento al espacio de direccionamiento del proceso.

```
#include <sys/shm.h>
char *shmat (shmid, *shmadr, shmflag);

int shmid;          /* identificador del segmento */
char *shmadr;       /* dirección de enlace */
int flag;           /* opción de conexión */
```

`shmid`: identificador del área de memoria compartida
`shmaddr`: dirección donde se mapea el área de memoria compartida. Si es NULL, el núcleo intenta encontrar una zona no mapeada.
`shmflg`: se indica el flag SHM_RDONLY si es solo para lectura.

Si la llamada a esta función funciona correctamente, devolverá un puntero a la dirección virtual a la que está conectado el segmento. Esta dirección puede coincidir o no con adr, dependiendo de la decisión que tome el kernel. El problema principal que se plantea es el de la elección de la dirección, puesto que, no puede entrar en conflicto con direcciones ya utilizadas, o que impida el aumento del tamaño de la zona de datos y el de la pila. Por ello, si se desea asegurar la portabilidad de la aplicación es recomendable dejar al kernel la elección de comienzo del segmento, para ello basta con pasarle un puntero NULL como segundo parámetro (valor entero 0). Si por algún motivo falla la llamada a esta función, devolverá –1.

Una vez que el segmento esta unido al espacio de direcciones virtuales del proceso, el acceso a él se realiza a través de punteros, como con cualquier otra memoria dinámica de datos asignada al programa.

Una vez que ya no se necesita más acceder al segmento de memoria compartida, el proceso debe realizar el desenlace. El desenlace no es lo mismo que la eliminación del segmento desde el núcleo. El segmento de memoria compartida se elimina sólo en el caso de que no queden procesos enlazados. Para el desenlace se utiliza el servicio `shmdt`.

La función *shmdt*[5] efectúa el desenlace del segmento previamente conectado.

```
#include <sys/shm.h>
int shmdt (*shmadr);
char *shmadr; /* dirección de conexión del segmento a desenlazar */
```

Su único parámetro shmaddr, es la dirección virtual del segmento que se desea desconectar (devuelta por la llamada a la función shmat). Es lógico que después de ejecutarse esta función, dicha dirección se convierta en una dirección ilegal del proceso. Esto no quiere decir que su contenido se borre, simplemente que se deja de tener acceso a la memoria reservada.

Si la llamada se efectúa satisfactoriamente, la función devolverá el valor 0, y en caso de que se produzca algún fallo devolverá –1.

4 http://pubs.opengroup.org/onlinepubs/009695399/functions/shmat.html
5 http://pubs.opengroup.org/onlinepubs/009695399/functions/shmdt.html

A continuación se muestra el código comentado de dos programas que utilizan memoria compartida, **demo1.c** y **demo2.c** . El primero, crea una zona de memoria para compartir con el segundo programa. Ejecute los dos programas al mismo tiempo en dos consolas distintas. Observe y analice los resultados.

4 Ejercicios prácticos

4.1 Ejercicio 1

Como se comentó en la práctica 1, los procesos a diferencia de los hilos, no comparten el mismo espacio de memoria, por lo que si queremos que accedan a las mismas variables en memoria, estos deben compartirla. Realice un programa que expanda N procesos hijos. Cada hijo debe compartir una variable denominada contador, que debe estar inicializada a cero. Esta variable debe ser incrementada por cada hilo 100 veces. Imprima la variable una vez finalicen los hilos y analice el resultado obtenido. Un resultado previsible para 3 procesos sería 300, así ha sido?

4.2 Ejercicio 2

Modificar el programa anterior para que el incremento de la variable contador se haga dentro de una zona de exclusión mutua para evitar así las posibles inconsistencias en los incrementos de la variable. Utilice para ello semáforos generales tipo mutex.

Nota: tenga en cuenta que el semáforo también deberá ser una variable compartida entre los distintos procesos.