UNIVERSIDAD DE CÓRDOBA
ESCUELA POLITÉCNICA SUPERIOR
DEPARTAMENTO DE INFORMÁTICA Y ANÁLISIS NUMÉRICO

ASIGNATURA ***SISTEMAS OPERATIVOS***

2º DE GRADO EN INGENIERÍA INFORMÁTICA

PRÁCTICA 4

Comunicación y sincronización entre procesos mediante semáforos y memoria compartida

Juan Carlos Fernández Caballero
jfcaballero@uco.es

Índice de contenido

1 Objetivo de la práctica

La presente práctica persigue instruir al alumnado con la comunicación inter-procesos utilizando el método de memoria compartida. En una 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 biblioteca ***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* en:

http://pubs.opengroup.org/onlinepubs/9699919799/

y la biblioteca GNU C (*glibc*) en:

http://www.gnu.org/software/libc/libc.html

3 Conceptos sobre memoria compartida

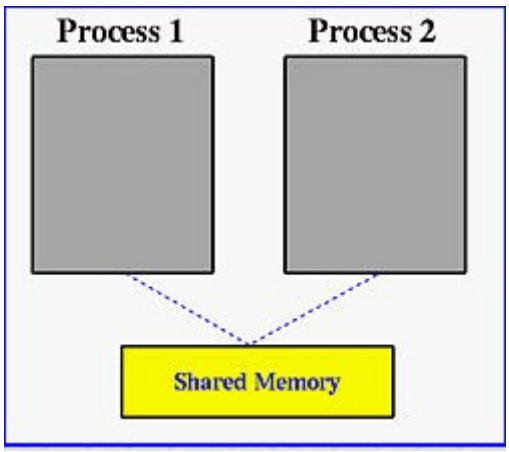
Una forma eficaz que tienen los procesos para comunicarse consiste en **compartir una zona de memoria** tal que, para enviar datos de un proceso a otro sólo se han de escribir en dicha memoria y automáticamente estos datos estarán disponibles para cualquier otro proceso. La utilización de este espacio 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 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. Por tanto, un proceso no puede acceder a la zona de memoria de otro proceso directamente.

GNU/Linux y su implementación POSIX 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**, 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 un conjunto de mecanismos de sincronización**, de forma que **se evite las típicas condiciones de carrera 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.

3.1 Claves IPC (ftok())

Antes de comenzar a estudiar las llamadas al sistema (*wrappers*) para la utilización de memoria compartida es necesario explicar que son las **claves IPC** (*InterProcess Communication*).

Una zona de memoria compartida es un objeto IPC que sirve para la comunicación entre procesos. Cada objeto IPC tiene un **identificador único** que se usa dentro del núcleo para identificar de forma única a ese objeto.

Para obtener el identificador único correspondiente a un objeto IPC se debe utilizar una ***clave***. Esta **clave debe ser conocida por todos los procesos que utilizan el objeto**. La clave puede ser estática, incluyendo su código en la propia aplicación o **generarla utilizando la función *ftok()***[1], cuyo prototipo es el siguiente:

```
#include <sys/ipc.h>
key_t   ftok  (char *pathname, int id);
```

- **pathname**: ruta de un nombre de un fichero existente en el sistema y accesible. Se puede indicar entre comillas dobles el directorio actual con un punto (“.”).
- **id**: variable entera o carácter que no debe ser 0.

1 http://pubs.opengroup.org/onlinepubs/9699919799/functions/ftok.html

Ejemplos:

key_t clave; *clave = ftok(“.”,’S’);*	*key_t clave;* *clave = ftok (“.”, 2);*	*key_t clave;* *clave = ftok (“/bin/ls”, 2);*	*key_t clave;* *clave = ftok (“/dev/null”, 2);*

La llamada a esta función devuelve una **variable de tipo *key_t***, que se corresponde con una **clave** que tendrá siempre el mismo valor cuando se reciben los mismos argumentos por parte de *ftok()*. Esta clave será empleada después en la llamada a la función *shmget()*. Si hay **error devuelve (-1) y establece valor en *errno***.

3.2 Petición de un segmento de memoria compartida (shmget())

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_t key,  size_t size,  int shmflg);
```

- **key**: El parámetro *key* es la clave IPC o llave de acceso, tal y como se explicó anteriormente, es decir, la clave del segmento de memoria compartida creada mediante *ftok()*. **La clave debe ser la misma para todos los procesos que quieran usar esta zona de memoria.**
- **size**: El segundo parámetro, *size*, indica el tamaño en bytes de la zona de memoria compartida que se desea crear. Este espacio es fijo durante su utilización. Se suele utilizar la opción de ***sizeof (tipo_variable)*** para que se tome el tamaño del tipo de dato que habrá en la memoria compartida.
- ***shmflg**:* El último parámetro *shmflg*, 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. Cuando se esté utilizando esta llamada al sistema para **acceder a una memoria que ya se ha creado** previamente, este argumento tomará el **valor 0**, pero *shmflg* puede tomar los siguientes valores:
  - ***IPC_CREAT***: Crea un segmento de memoria si no existe.
  - *IPC_EXCL*: Al usarlo junto con *IPC_CREAT*, falla si el segmento ya existe y no se ha eliminado (*IPC_CREAT | IPC_EXCL*).
  - SHM_R: Para lectura.
  - SHM_W: Para escritura.
  - SHM_R | M_W: Para lectura-escritura.
  - Podemos poner lo que deseemos para el usuario, el grupo del usuario y resto de usuarios del sistema:

    Ejemplo: *IPC_CREAT | 0660* indica permisos de lectura y escritura para el usuario y el grupo al que pertenece el usuario propietario del proceso.

*shmget() d*evuelve el **identificador del segmento de memoria compartida** si hubo éxito o devuelve **(–1) en caso de error**, por lo que habría que consultar la variable *errno* (consulte

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

OpengGroup). **El identificador del segmento que devuelve esta función es heredado por todos los procesos descendientes (*fork()*) del que llama a esta función. Se podría usar dependiendo del problema a resolver.**

El siguiente código muestra un ejemplo de 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 sin usar ftok() */

int shmid;    /* identificador del nuevo segmento de memoria compartida */

if( ( shmid = shmget (LLAVE, 4096, IPC_CREAT  | IPC_EXCL  | 0600) ) == -1)
{
    /* Error al crear o habilitar el segmento de memoria compartida. Tratamiento del error. */
}
...
```

3.3 Conexión del segmento de memoria compartida (shmat())

Para que un proceso pueda acceder a esa memoria compartida previamente creada, será necesario que **alguna variable del proceso "apunte" a esa zona de memoria que no pertenece a su espacio de direccionamiento**. 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*[3] realiza la conexión del segmento al espacio de direccionamiento del proceso:

```
#include <sys/shm.h>
void * shmat (int shmid,   const void *shmaddr,   int shmflg);
```

- ***shmid***: identificador de la memoria compartida. Lo habremos obtenido con la llamada *shmget()*.

- ***shmaddr***: normalmente, valdrá **NULL que indicará al sistema operativo que busque esa zona de memoria en una zona de memoria libre**, no en una dirección absoluta.
  El problema principal que se plantea en la elección de la dirección es que no puede entrar en conflicto con direcciones ya utilizadas, o que impidan el aumento del tamaño de la zona de datos y pila de los procesos. 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.

- ***shmflg***: modo lectura, escritura o ambos. Un **0 es la opción por defecto**, que **significa lectura-escritura**. Si se indica el *flag SHM_RDONLY* si es solo para lectura (consulte *OpenGroup*).

3 http://pubs.opengroup.org/onlinepubs/009695399/functions/shmat.html

Si la llamada a esta función funciona correctamente **devolverá un puntero a la zona de memoria compartida**. 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.

3.4 Control de un segmento de memoria compartida (shmctl())

La función *shmctl*[4] proporciona información administrativa y de control sobre el segmento de memoria compartida que se especifique. En particular, se va a usar para **liberar la memoria compartida**. Su declaración es la siguiente:

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

- ***shmid***: identificador de la zona de memoria compartida. Lo habremos obtenido con la llamada *shmget()*.
- ***cmd:*** Es la operación que se quiere realizar. 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.
    - *PC_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. Normalmente lo que se usa es la opción ***IPC_RMID*** que marca esa zona de memoria para ser liberada cuando ya no haya ningún proceso vinculado a ella.
- ***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*. **Usaremos NULL** si no nos interesa guardar esa información.

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

El siguiente 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. */
}
```

4 http://pubs.opengroup.org/onlinepubs/009695399/functions/shmctl.html

3.5 Desenlace de un segmento de memoria compartida (shmdt())

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 la llamada a la función *shmdt*[5] efectúa el desenlace del segmento previamente conectado:

```
#include <sys/shm.h>
int shmdt (const void *shmaddr);
```

- *shmaddr:* el parámetro *shmaddr* es la dirección del segmento que se desea desconectar (valor devuelto 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 (recuerde que la memoria compartida reservada se encuentra fuera del espacio del proceso).

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

A continuación se muestra el código 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, primero arrancando demo1 y luego demo2. Observe y analice los resultados.

3.6 Combinación de semáforos con memoria compartida

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, siendo la técnica de memoria compartida que se acaba de explicar una buena solución para ello.

Por otro lado, si varios procesos van a acceder a una misma zona de memoria compartida se pueden ocasionar **condiciones de carrera**, por lo que habría que habría que acceder a ella en **exclusión mutua**. Para ello se podrían utilizar los **semáforos generales** estudiados en las prácticas anteriores.

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

Tenga en cuenta que **el semáforo también deberá ser una variable compartida entre los distintos procesos, los semáforos no se heredan** por los procesos hijos.

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, int value);
```

- **sem**: Puntero a parámetro de tipo *sem_t* que identifica el semáforo.
- **pshared**: Entero que determina si el semáforo sólo puede ser utilizado por hilos del mismo proceso que inicia el semáforo, en cuyo caso pondremos como parámetro el **valor 0**,

  o se puede utilizar para sincronizar procesos, en cuyo caso pondremos un **valor de 1**.

  Fíjese en el segundo parámetro de la inicialización de un semáforo binario para usarlo como memoria compartida entre procesos:

```
sem_init(mutex,1,1);
```

- El segundo parámetro esta puesto a **1**, en vez de a 0 como hasta ahora, lo que significa que **el semáforo se va a utilizar entre procesos.**

4 Ejercicios prácticos

4.1 Ejercicio 1

Realice un programa que expanda con *fork() N* procesos hijos. Cada hijo debe compartir una variable denominada *contador*, que debe estar inicializada a 0. Esta variable, que debe ser reservada como memoria compartida (no sirve como variable global) debe ser incrementada por cada hijo *100 veces*. Imprima la variable una vez finalicen los hilos y analice el resultado obtenido.

Un resultado previsible sería 300. ¿Ha sido así?

4.2 Ejercicio 2

Modificar el programa anterior para que el incremento de la variable *contador* se haga en *exclusión mutua*, para evitar así las posibles inconsistencias en los incrementos de la variable. Utilice para ello los *semáforos* generales estudiados en las prácticas anteriores.