Prácticas de Sistemas Operativos

Memoria Compartida

Javier J. Gutiérrez

Índice

Este boletín no entra dentro del temario de la asignatura y, por tanto, no se preguntará en ningún examen.

1. Introducción

La memoria compartida es uno de los mecanismos agrupados bajo el nombre de Inter. Process Comunication (IPC), junto con semáforos y colas de mensajes (FIFO)que tienen disponibles los sistemas Unix basados en System V, entre ellos el sistema SunOS de Murillo y cualquier Linux. En este boletín se va a describir brevemente el funcionamiento de la memoria compartida. Como se verá, el mecanismo es muy similar al funcionamiento de los semáforos y de las colas de mensajes.

Mediante memoria compartida, como su nombre indica, podemos crear zonas de memoria compartidas por varios procesos. De este modo los cambios que un proceso realice a los valores almacenados en memoria compartida son visibles para los demás procesos que utilicen esa misma memoria compartida. Por este motivo, en algunos casos será necesario garantizar el acceso en exclusiva para evitar inconsistencias en los datos mediante mecanismos ya vistos como señales o semáforos. Por ejemplo, si dos procesos acceden a la vez a este área y intentan modificarla seguramente la modificación no será correcta.

Este recurso IPC es el mas rápido de los tres, ya que una vez conectados no necesitamos hacer mas llamadas al sistema, ni interactuar con el núcleo; directamente escribimos o leemos gracias un puntero que referencia a la memoria compartida.

A continuación se muestra una tabla comparativa de los tres mecanismos.

	Semáforos	**Colas mensajes**	**Memoria compartida**
Archivo de cabecera	`sys/sem.h`	`ys/msg.h`	`sys/shm.h`
Creación / Obtención	`semget(…)`	`msgget(…)`	`shmget(…)`
Control	`semctl(…)`	`msgctl(…)`	`shmctl(…)`
Operaciones	Subir / bajar	Poner / quitar mensaje	Vincular / desvincular

Algunos sistemas, como SunOS y Linux, permiten trabajar con memoria compartida vinculada a un fichero (funciones `map` y `munmap`). Este tipo de memoria no se explica en este boletín.

2. Creación de zonas de memoria compartida

Para crear zonas de memoria compartida utilizaremos la función `shmget` de una manera similar a los semáforos y colas de mensajes.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget (key_t clave, int tam, int flag);
```

Devuelve -1 en caso de error y un identificador de la zona de memoria compartida en caso de éxito. Ese identificador lo utilizaremos para trabajar con la zona de memoria compartida.

key_t clave – clave de la zona de memoria compartida.

int tam – tamaño en bytes de la zona de memoria compartida si queremos crearla. Si queremos acceder a un área ya creada, el tamaño será 0.

int flag – permisos. Se indican de la misma manera que los semáforos y colas de mensajes.

Ejemplo – *El siguiente fragmento de código crea una zona de memoria compartida del tamaño de una variable entera. La zona de memoria tendrá permisos de escritura y lectura sólo para el propietario.*

```
int shmid;
shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | IPC_EXCL
                                          | S_IRUSR | S_IWUSR);
if (shmid == -1)
    perror("Error creando memoria compartida.");
```

3. Vinculación de memoria compartida

Ya hemos creado la memoria compartida, pero para utilizarla necesitamos saber su dirección (observe que `shmget` no nos la indica). Para utilizar la memoria compartida debemos antes vincularla con alguna variable de nuestro código. De esta manera, siempre que usemos la variable vinculada estaremos utilizando la variable compartida. Para establecer un vínculo utilizamos la función `shmat`.

```
void *shmat(int shmid, const void *shmaddr, int shmflag)
```

Devuelve la dirección de memoria de comienzo de la memoria compartida o -1 si hubo algún error.

int shmid – identificador de la memoria compartida obtenido con `shmget`.

const void *shmaddr – Dirección concreta donde reside la memoria compartida. No lo vamos a utilizar por lo que su valor siempre será `NULL`.

int shmflag – permisos. Por ejemplo. Aunque hayamos obtenido una zona de memoria con permiso para escritura o lectura, podemos vincularla a una variable para solo lectura. Si no queremos cambiar los permisos usamos 0.

- `SHM_RND`: Indica que la dirección de la memoria debe redondearse a la dirección de la página de memoria más cercana. No la utilizaremos.
- `SHM_RDONLY`: Indica que la memoria compartida será de solo lectura.

Ejemplo *– El siguiente fragmento de código recupera la dirección de la zona de memoria compartida obtenida en el ejemplo anterior y coloca en ella el valor 10. Observe que, como la dirección de memora corresponde con un valor entero, la almacenamos en un puntero a entero. Observe también como se realiza adecuadamente la comprobación de errores.*

```
int *entero;
entero = (int *)shmat(shmid, NULL, 0);
if (entero == (int *)-1) {
    perror("Obteniendo dirección de memoria compartida");
    return -1;
}
(*entero)=10;
```

Cuando ya no vamos a utilizar más la variable vinculada, hemos de desvincularla antes de poder eliminar la memoria compartida. Para ello utilizamos la función `shdmt.`

```
int shmdt(const void *shmaddr)
```

Devuelve -1 si hubo algún error, otro valor en caso contrario. Observe que el -1 devuelto no es de tipo entero sino de tipo puntero a entero, es decir, el puntero apunta a la posición de memoria -1.

const void *shmaddr – Variable vinculada a la memoria compartida.

Ejemplo *– El siguiente fragmento desvincula la memoria compartida vinculada en el ejemplo anterior.*

```
r = shmdt(entero);
if (r == -1)
    perror("Error desvinculando memoria compartida");
```

Ojo, esta función sólo libera el vínculo, pero no elimina la zona de memoria compartida. Además los vínculos se pierden al llamar a las funciones `exit` o `exec` (en sistemas Linux, en otras variantes de Unix esto puede no ser cierto).

4. Borrado de memoria compartida

Como los demás recursos IPC, la memoria compartida reservada seguirá estando presente en el sistema hasta que no la borremos explícitamente. Para ello utilizamos la función `memctl` de una forma muy similar a como borramos grupos de semáforos o colas de procesos.

```
int shmctl(int shmid,int cmd, struct shmid_ds *buf)
```

Devuelve -1 si error, u otro valor en caso contrario.
int shmid – identificador de la memoria compartida obtenido con `shmget`.
int cmd - alguna de las siguientes constantes:

- IPC_STAT: Nos rellena la estructura `shmid_ds()` con los datos que maneja el núcleo en lo referente a la zona de memoria compartida.
- IPC_SET: Lo contrario de lo anterior establece la estructura que le pasamos como parámetro en el kernel.
- IPC_RMID: Borra el recurso.

Algunos sistemas, como Linux o SunOS, soportan también las constantes `SHM_LOCK` y `SHM_UNLOCK` para bloquear o desbloquear el acceso a memoria compartida.

struct sumid_ds *buf - para el comando de borrado no es necesario el tercer argumento, por lo que utilizaremos NULL.

Ejemplo *– La siguiente línea borra la zona de memoria compartida que hemos utilizado en los ejemplos anteriores.*

```
r = shmctl(shmid, IPC_RMID, NULL);
if (r == -1)
    perror("Error desvinculando memoria compartida");
```

Ojo, la operación de borrado sólo es efectiva si ningún proceso tiene vinculada la memoria compartida. Si aún existen vínculos a la memoria compartida, la operación de borrado se pospone hasta que desaparezca el último vínculo.

La estructura `shmid` se muestra a continuación

```
struct shmid_ds{
   struct ipc_perm shm_perm; //Permisos
   int shm_segsz; //Tamaño del área
   time_t shm_atime; //Hora del último smhmat
   time_t shm_dtime; //Hora del último shmdt()
   time_t shm_ctime; //Hora del último shmctl()
   unsigned short shm_cpid; //Pid del creador
   unsigned short shm_lpid; //Pid del ultimo que hizo shmctl()
   short shm_nattach; //Num. de procesos conectados
}
```

La estructura `ipc_perm` se muestra a continuación:

```
struct ipc_perm {
    ushort cuid;
    ushort cgid;
    ushort uid;
    ushort gid;
    ushort mode;
    ushort seq;
    key_t key;
};
```

5. Ejercicios

1. Construya una librería de funciones que permita trabajar con una variable de tipo entero.

2. A partir del ejercicio anterior, construya una Liberia de funciones que permita trabajar con matrices de enteros de cualquier número de elementos.