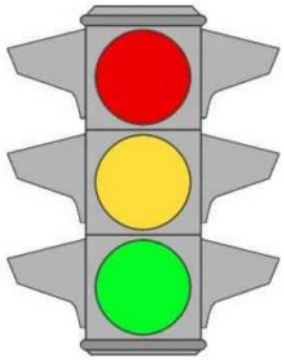
Implementación de semáforos POSIX en C y librería de servicio

C Prácticas, UNIDAD 3 - Sistema de Procesos, Unidad 3 Tema 5



En esta entrada mostramos cómo se implementan realmente los semáforos en C y en linux (POSIX), y cómo desarrollar una librería para facilitar su uso, con las funciones originales ofrecidas por el S.O. En otra entrada pondremos un ejemplo del uso de los semáforos para sincronizar a los padres e hijos.

Un semáforo en Unix / Linux se compone de algunos elementos más que los que hemos visto en teoría. Un semáforo POSIX, se encuentran definido en <sys/sem.h> habitualmente y en la realidad se componen de los siguientes elementos:

- El valor del semáforo
- El identificador del último proceso que manipuló el semáforo
- El número de procesos que hay esperando a que el valor del semáforo se incremente

- El número de procesos que hay esperando a que el semáforo tome el valor 0.

Las llamadas relacionadas con los semáforos son:

- semget : Para crear un semáforo o habilitar el acceso a uno ya existente
- semctl : Para realizar operaciones de control, inicialización y destrucción.
- semop : Para realizar operaciones sobre el semáforo (wait y signal)

Nosotros vamos a encapsular en una librería propia una serie de funciones que harán uso de las que hemos mencionado para que el uso de los semáforos en el código sea lo más parecido a lo que hemos visto en teoría.

Nuestras funciones serán, las siguientes:

- sCreate : Para crear el semáforo pasándole el valor de inicio
- sWait : Para hacer la operación wait sobre el semáforo
- sSignal : Para hacer la operación signal sobre el semáforo
- sDestroy: Para destruir el semaforo. Los semáforos son ficheros que quedan en el sistema si no se destruyen.

Para crear estas funciones debemos entender primero las funciones originales. Una vez sepamos lo que el S.O. nos ofrece montaremos nuestras funciones.

Para trabajar con semáforos tenemos que:

1. Pedir la clave del semáforo, que es un identificador de recurso compartido.
2. Crear el semáforo usando la clave.
3. Inicializar el semáforo
4. Usar el semáforo
5. Destruir el semáforo

Petición de la clave del semáforo

ftok es una función que permite obtener un identificador para un recurso compartido, en este caso un semáforo, pero también se usa para colas de mensajes y memoria compartida.

```
key_t ftok(char *, int)
```

a la que se suministra como primer parámetro el nombre y path de un fichero cualquiera que exista y al que se tenga acceso y como segundo un entero cualquiera.

Importante: *Todos los procesos que quieran compartir el semáforo, deben suministrar el mismo fichero y el mismo entero.*

Creación del semáforo

Mediante semget vamos a poder crear o acceder a un conjunto de semáforos (un array de semáforos) unidos bajo un identificador común.

Semget tiene la siguiente declaración:

```
int semget(key_t key, int nsems, int semflg);
```

Si la llamada funciona devolverá un identificador con el que podremos acceder a los semáforos. Si falla, semget devuelve -1 y en errno estará el código de error producido.

1. key es la llave que indica a qué grupo de semáforos queremos acceder. Esta llave se puede crear mediante una llamada a ftok. Si por el contrario pasamos IPC_PRIVATE, la llamada crea un nuevo identificador sujeto a la disponibilidad de entradas libres que el nucleo dispone.
2. nsems es el total de semáforos que vamos a crear bajo el mismo identificador devuelto por semget.
3. semflg es una máscara de bits que indica el modo de adquisición del identificador. Si el bit IPC_CREAT está activo, se creará. Los 9 bits menos significativos de la máscara son los permisos del semáforo.

Estos flags permiten poner los permisos de acceso a los semáforos, similares a los ficheros, de lectura y escritura para el usuario, grupo y otros. También lleva unos modificadores para la obtención del semáforo.

Podemos pues usar IPC_CREAT | 0644 que indica permiso de lectura y escritura para el propietario y de lectura para el resto (grupo y otros) y que los semáforos se creen si no lo están ya al llamar a semget(). El cero delante del 644 indica al

compilador que el número es octal y los permisos se asignan correctamente.

La función semget() nos devuelve el identificador del array de semáforos.

El identificador devuelto por semget es heredado por los procesos descendientes del actual. Por tanto los hijos ya tendrán creado el semáforo si lo crea el padre antes del fork.

Inicializar el semáforo

Con semctl vamos a poder acceder a la información administrativa y de control que el núcleo ofrece sobre el semáforo. Es decir, no solo inicializarlo, sino más cosas, aunque nosotros veamos sólo como inicializarlo.

La declaración de semctl es la siguiente:

```
int semctl (semid, semnum, cmd, arg);
int semid, semnum, cmd;
union semnum{
   int val;
   struct semid_ds *buf;
   ushort *array;
}arg;
```

- semctl actuará sobre el conjunto de semáforos identificados por semid, que es lo que devuelve la función semget que hemos visto antes.
- semnum indica cuál es el semáforo al que queremos acceder, es decir, el índice del array de semáforos. Tener en cuenta que empieza por 0.
- cmd indica qué operación se va ha hacer en el semáforo, entre las que disponemos de:
  - GETVAL para leer el valor del semáforo
  - SETVAL para inicializar el semáforo al valor especificado en el parámetro arg
  - GETPID para leer el PID del último proceso que actuó sobre el semáforo
  - GETNCNT para leer el número de procesos que hay esperando a que se incremente el semáforo
  - GETZCNT para leer el número de procesos que hay esperando a que el semáforo se ponga a 0
  - GETALL permite leer el valor de todos los semáforos asociados al identificador. El valor se almacena en arg, que deberá ser un puntero a un array de enteros.
  - SETALL permite inicializar el valor de todos los semáforos asociados al identificador

- IPC_RMID permite la destrucción del semáforo

Si la función se realiza con éxito devolverá un número que depende, como hemos visto, del valor de cmd cuando este es GETVAL, GETNCNT, GETZCNT o GETPID, para otros valores de cmd devuelve 0 si éxito y -1 si error.

Operar con el semáforo

Para realizar operaciones con el semáforo utilizaremos la llamada al sistema semop.

```
int semop( int semid, struct sembuf *sops, int nsops);
```

- semop realiza operaciones atómicas sobre los semáforos que hay asociados bajo el identificador del array.
- sops es un puntero a un array de estructuras que indican las operaciones que vamos a hacer. Para cada semáforo tendremos una estructura que indica que operación hacemos sobre él.
- nsops es el total de elementos que tiene el array de operaciones sops.

Cada elemento del array es una estructura sembuf que se define como sigue:

```
struct sembuf{
  ushrot sem_num; //Número del semáforo (indice del array)
  short sem_op; //Operación, incrementar o decrementar
  short sem_flg; //máscara de bits
```

- sem_num es el número del semáforo, es decir, el índice del array de semáforos. Empieza por 0
- sem_op es la operación a realizar sobre el semáforo.
  - Si sem_op < 0 el semáforo se decrementa (wait)
  - Si sem_op = 0 el semáforo se queda como está
  - Si sem_op > 0 el semáforo se incrementa (signal)

Las operaciones signal siempre terminan con éxito puesto que se puede incrementar el valor del semáforo hasta donde se quiera. Sin embargo, en la implementación de los semáforos, no pueden tomar valores negativos, es decir, si el valor del semáforo es 2 no se puede poner sem_op a valores mayores que 2. ¿Que pasa si pasamos valores que hacen negativo el valor del semáforo? Dependerá de sem_flg

- sem_flg es una máscara de bits tal que:
  - si ponemos IPC_NOWAIT el S.O. devuelve el control al proceso que hace la llamada. Por defecto es IPC_WAIT, que causa el bloqueo del proceso que intenta poner a negativo el valor del semáforo.
  - El valor por defecto de sem_flg es 0, que es el que usaremos en nuestros programas.

Destruir el semáforo

Para destruir el semáforo tenemos que hacer uso de la llamada al sistema semctl.

```
int semctl(int semid, int semnum, int cmd, ...);
```

Este es el prototiopo de la llamada al sistema.

El parámetro semid será el que se nos retornó al crear el semáforo.
El parámetro semnum será obviado para la operación de eliminación
El parámetro cmd que hay que utilizar es IPC_RMID

Con esto, para destruir un semáforo haríamos:

```
ret=semctl(semID, 0, IPC_RMID);
if (ret == -1 ) {
    perror("Error al obtener el semID: ");
    exit(1);
}
```

Creación de nuestra librería de semáforos

Una vez que hemos conocido lo que el sistema operativo nos proporciona, podemos crear una librería sem.h para encapsular todas estas funciones y dejar únicamente aquellas que nos interesa en la forma en que nos interesa. La explicamos a continuación.

sem.h

La librería podría ser como esta:

```
//Creamos un semáforo con un valor inicial
int sCreate(int seed, int value);

//Operaciones de incremento y decremento del semáforo
void sWait(int semID);
void sSignal(int semID);
```

sem.c

La implementación de esta librería, en su fichero sem.c, tendremos:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#include <errno.h>
#include <stdlib.h>

#define PERMISOS 0644

typedef union semun {
    int val;               /* used for SETVAL only */
    struct semid_ds *buf; /* for IPC_STAT and IPC_SET */
    ushort *array;         /* used for GETALL and SETALL */
    } semun;


int sGet(int seed)
{
 key_t semKey = ftok("/bin/ls", seed);
    if (semKey == -1 ) { perror("Error al crear el token: "); exit(-1);}
 int semID = semget(semKey,1,IPC_CREAT | PERMISOS);
    if (semID == -1 ) { perror("Error al obtener el semID: "); exit(-1);}
 return semID;
}

void sSet(int semID, int value)
{
 semctl(semID, 0, SETVAL, value);
}

int sCreate(int seed, int value)
{
 int semID = sGet(seed);
 sSet(semID, value);
```

```
 return semID;
}

void sWait(int semID)
{
 struct sembuf op_Wait [] =  { 0, -1, 0 }; // Decrementa en 1 el semáforo
 semop ( semID, op_Wait, 1 );
}

void sSignal(int semID)
{
 struct sembuf op_Signal [] = { 0, 1, 0 }; // Incrementa en 1 el semáforo
 semop ( semID, op_Signal, 1 );
}

void sDestroy(int semID)
{
    semun dummy;
    int ret;

    ret=semctl(semID, 0, IPC_RMID, dummy);
    if (ret == -1 ) { perror("Error al destruir el semaforo: "); exit(-1);}
}
```

En la implementación tenemos definidas las funciones sGet y sSet que no vamos a usar en el código de nuestros ejemplos, pero que si son necesarias para la función sCreate que creamos.

- sGet utiliza ftok para pedir un identificador de recurso compartido, basado en un fichero existente. Para ello usamos el fichero /bin/ls que seguro que existe. Llamamos a semget con la clave obtenida y creamos el array de 1 único semáforo con los permisos definidos.
- sSet utiliza semctl para inicializar el semáforo al valor pasado.
- sCreate utiliza estas dos funciones para crear e inicializar el semáforo en una única llamada.
- sWait se usará para decrementar en 1 el semáforo, el único que existe, el de indice 0. Los flags también los pone a 0 por lo que es bloqueante.
- sSignal se usará para incrementar en 1 el semáforo cuyo indice es 0, los flags también a su valor por defecto.
- sDestroy se usará para destruir el semáforo y que no quede en el sistemas. Es sistema puede intentar reutilizarlo y esto puede par problemas si ha ocurrido un error con este semáforo y no se destruyó apropiadamente.

Estas funciones nos abstraen de la complejidad del uso de las funciones nativas, pero a costa de perder posibilidades, hay que tener eso en cuenta. Para nuestro uso es suficiente.

Una vez que tenemos todo definido podemos ver cómo se usan los semáforos en un ejemplo que usa la librería que hemos definido.

El ejemplo lo podemos ver en Sincronización Padre-Hijos con semáforos.

← Entrada anterior

Entrada siguiente →

Escuela Politécnica Superior de Elche

Universidad Miguel Hernández

Miguel Onofre Martínez Rach