Semáforos en C bajo Linux

A veces es necesario que dos o más procesos o hilos (*threads*) **accedan a un recurso común** como ser:

- Escribir en un mismo fichero.
- Leer la misma zona de memoria.
- Escribir en la misma pantalla.
- etc.

El **problema** es que, si lo hacen simultáneamente y de forma incontrolada, pueden "machacar" el uno la operación del otro (y dejar el fichero o la memoria con un contenido inservible o la pantalla ilegible).

Para evitar este problema, están los semáforos. **Un semáforo da acceso al recurso a uno de los procesos y se lo niega a los demás** mientras el primero no termine. Los semáforos, junto con la memoria compartida y las colas de mensajes, son los recursos compartidos que suministra UNIX para comunicación entre procesos (IPC).

El funcionamiento del semáforo es como el de una variable “contador”. Imaginar que el semáforo controla un fichero y que **inicialmente** tiene el valor 1 → está "VERDE". Cuando un proceso quiere acceder al fichero, primero debe decrementar el semáforo. El contador queda a 0 y como no es negativo, deja que el proceso siga su ejecución y, por tanto, acceda al fichero.

Ahora un segundo proceso lo intenta y para ello también decrementa el contador. Esta vez el contador se pone a -1 y como es negativo, el semáforo se encarga de que **el proceso quede "bloqueado" y "dormido"** en una cola de espera. Este segundo proceso no continuará por tanto su ejecución y no accederá al fichero.

Suponer ahora que el primer proceso termina de escribir el fichero. Al acabar con el fichero debe incrementar el contador del semáforo. Al hacerlo, este contador se pone a 0. Como no es negativo, el semáforo se encarga de mirar el la cola de procesos pendientes y "desbloquear" al primer proceso de dicha cola. Con ello, el segundo proceso que quería acceder al fichero continua su ejecución y accede al fichero.

Cuando este proceso también termine con el fichero, incrementa el contador y el semáforo vuelve a ponerse a 1, a estar "verde".

Es posible hacer que el valor inicial del semáforo sea, por ejemplo, 3, con lo que pasarán los tres primeros procesos que lo intenten. Pueden a su vez quedar muchos procesos encolados simultáneamente, con lo que el contador quedará con un valor negativo grande. Cada vez que un proceso incremente el contador (libere el recurso común), el primer proceso encolado despertará. Los demás seguirán dormidos.

Como se puede ver, el proceso de los semáforos requiere colaboración de los procesos. Un proceso debe decrementar el contador antes de acceder al fichero e incrementarlo cuando termine. Si los procesos no siguen este "protocolo" (y pueden no hacerlo), el semáforo no sirve de nada.

Código de los semáforos

Antes de nada, que quede claro que únicamente se pretende dar una idea sencilla de cómo funcionan los semáforos. No se detallan aquí todas las opciones de las funciones a utilizar ni se explican todas las posibilidades. Tampoco es posible garantizar que la sintaxis de las funciones y de los parámetros sea correcta al 100%, aunque sí se suministran dos fuentes de ejemplo que compilan y funcionan. Hay, por tanto, que leer este texto con intención de hacer únicamente un uso sencillo de semáforos.

Para utilizar los semáforos en un programa, deben seguirse los siguientes pasos:

1) Obtener una clave de semáforo

En general, una clave de recurso compartido, ya que la función que sirve para obtener dicha clave también vale para memoria compartida y colas de mensajes. Para ello se utiliza la función key_tftok(char *, int) a la que se suministra como primer parámetro el nombre y *path* de un fichero cualquiera que exista y como segundo un entero cualquiera. Todos los procesos que quieran compartir el semáforo deben suministrar el mismo fichero y el mismo entero. En los programas de ejemplo se utilizan "/bin/ls" y el entero 33.

2) Obtener un array de semáforos

La función intsemget (key_t, int, int) permite obtener un *array* de semáforos. Se le pasa como primer parámetro la clave obtenida en el paso anterior, el segundo parámetro es la cantidad de semáforos que se quiere y el tercer parámetro son *flags*. 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. En el ejemplo se pondrá **0600 | IPC_CREATE,** que indica permiso de lectura y escritura para el propietario y que **los semáforos se creen si no lo están al llamar a semget().** Es importante el 0 delante del 600, así el compilador de C interpretará el número en octal y pondrá correctamente los permisos. La función semget() devuelve un identificador del *array* de semáforos.

3) Uno de los procesos debe inicializar el semáforo

La función para utilizar es intsemctl (int, int, int, int). El primer parámetro es el identificador del *array* de semáforos obtenido anteriormente, el segundo parámetro es el índice del semáforo a inicializar dentro del *array* de semáforos obtenido. Si sólo se ha pedido uno, el segundo parámetro será 0. El tercer parámetro indica qué se quiere hacer con el semáforo. En función de su valor, los siguientes parámetros serán una cosa u otra. En el ejemplo, que se desea inicializar el semáforo, el valor del tercer parámetro es SETVAL. El cuarto parámetro, aunque está definido como un entero, en realidad admite una unión bastante liada. Para no complicarse, bastará pasar como cuarto parámetro un 1 si se desea el semáforo en "verde" o un 0 si se lo quiere en "rojo". En el código de ejemplo se utiliza la unión, pero el resultado es el mismo.

4) Ya está todo preparado, ahora sólo queda usar los semáforos

El proceso que quiera acceder a un recurso común debe primero decrementar el semáforo. Para ello utilizará la función intsemop (int, strcutsembuf *, size_t). El primer parámetro es el identificador del array de semáforos obtenido con semget(). El segundo parámetro es un *array* de operaciones sobre el semáforo. Para decrementarlo, bastará con un *array* de una única posición. El tercer parámetro es el número de elementos en el *array*, es decir, 1. La estructura del segundo parámetro contiene tres campos:

a) short sem_num que es el índice del *array* del semáforo sobre el que se quiere actuar. En este caso, con un sólo semáforo, el índice será 0.
b) short sem_op que es el valor en el que se quiere decrementar el semáforo. En este caso, -1.
c) short sem_flg son *flags* que afectan a la operación. En este caso, para no complicarse, se pondrá 0.

Al realizar esta operación, si el semáforo se vuelve negativo, el proceso se quedará "bloqueado" hasta que alguien incremente el semáforo y lo haga, como mínimo, 0.

5) Cuando el proceso termine de usar el recurso común, debe incrementar el semáforo. La función a utilizar es la misma, pero poniendo 1 en el campo sem_op de la estructura strcutsembuf.

Se compilan por separado utilizando el comando GCC por consola. Para ejecutarlos, debe ejecutarse primero el proceso que inicializa el semáforo en una ventana o terminal delshell y luego el otro proceso en otra ventana delshell distinta.

Cuando el proceso tiene un bucle infinito para entrar en el semáforo. Escribe en pantalla cuando entra y cuando sale.

Cuando el proceso tiene un bucle de 1 a n en el que pone en verde el semáforo y espera un segundo. El resultado es que entra en el semáforo, queda "bloqueado" un segundo y sale del semáforo para volver a entrar en él y repetir el proceso n veces.

Puesto que las llamadas a la función semop() son bastante "engorrosas" por aquello de rellenar la estructura, en el código hay un par de funcionesespera_semaforo() y levanta_semaforo() que decrementan e incrementan el semáforo.

La función espera_semaforo() es una función “bloqueante”, recordar que, si el semáforo se vuelve negativo, el proceso quedará “bloqueado”.

```
#include <sys/ipc.h>
#include <sys/sem.h>
#define ROJO 0
#define VERDE 1
#define CLAVE_BASE 33
```

```
key_t creo_clave(int r_clave)
{
 // Igual que en cualquier recurso compartido (memoria compartida, semaforos
 // o colas) se obtien una clave a partir de un fichero existente cualquiera
 // y de un entero cualquiera. Todos los procesos que quieran compartir este
 // recurso, deben usar el mismo fichero y el mismo entero.
    key_t clave;
    clave = ftok ("/bin/ls", r_clave);
    if (clave == (key_t)-1)
```

```
    {
        printf("No puedo conseguir clave para memoria compartida\n");
        exit(0);
    }
    return clave;
}

//funcion que crea el semaforo
int creo_semaforo()
{
  key_t clave = creo_clave(CLAVE_BASE);
  int id_semaforo = semget(clave, 1, 0600|IPC_CREAT);
  //PRIMER PARAMETRO ES LA CLAVE, EL SEGUNDO CANT SEMAFORO, EL TERCERO 0600
LO UTILIZA CUALQUIERA, IPC ES CONSTANTE (VEA SEMAFORO)
  if(id_semaforo == -1)
  {
      printf("Error: no puedo crear semaforo\n");
      exit(0);
  }
  return id_semaforo;
}

//inicia el semaforo
void inicia_semaforo(int id_semaforo, int valor)
{
    semctl(id_semaforo, 0, SETVAL, valor);
}



//levanta el semaforo
void levanta_semaforo(int id_semaforo)
{
    struct sembuf operacion;
    printf("Levanta SEMAFORO \n");
    operacion.sem_num = 0;
    operacion.sem_op = 1; //incrementa el semaforo en 1
    operacion.sem_flg = 0;
    semop(id_semaforo,&operacion,1);
}

//espera semaforo
void espera_semaforo(int id_semaforo)
{
    struct sembuf operacion;
```

```
    printf("Espera SEMAFORO \n");
    operacion.sem_num = 0;
    operacion.sem_op = -1; //decrementa el semaforo en 1
    operacion.sem_flg = 0;
    semop(id_semaforo,&operacion,1);

}
```

Estructura de la función main() con semáforos:

```
int main()
{
    int i;
    int id_semaforo;

    id_semaforo = creo_semaforo();

    inicia_semaforo(id_semaforo, VERDE);

    while(1)
    {
        espera_semaforo(id_semaforo);
            printf("Seccion critica\n");
            sleep (1);
       levanta_semaforo(id_semaforo);
        sleep (10);
    }

    return 0;
}
```