*Departamento de Ingeniería Telemática*
*Escuela Superior de Ingenieros*
*Universidad de Sevilla*

# *PRÁCTICA 5. SEMÁFOROS*

## 1. OBJETIVO

El objetivo es comprender las llamadas al sistema en UNIX para la utilización de semáforos.

## 2. DESCRIPCIÓN

En los semáforos se hace la distinción entre:

- Semáforos sin nombre: Permiten sincronizar procesos emparentados que heredan el semáforo a través de la llamada *fork* (también a procesos ligeros que ejecutan dentro de un mismo proceso).

- Semáforos con nombre: El semáforo lleva asociado un nombre que sigue la convención de nombrado que se utiliza para ficheros. Este tipo se semáforos sirve para sincronizar procesos no emparentados.

En esta práctica se van a estudiar los semáforos con nombre.

Para la identificación de un semáforo se utiliza una variable de tipo sem_t.

### Creación y apertura de un semáforo con nombre

Para crear o abrir un semáforo con nombre se utiliza el servicio `sem_open`. El prototipo de la función que se utiliza para este servicio tiene dos modalidades, según se utilice para crear el semáforo o simplemente para abrir uno que existe.

Para crear un semáforo el prototipo que se utiliza es el siguiente:

```
sem_t *sem_open(char *name, int flag, mode_t mode, int val);
```

Para abrir un semáforo existente el prototipo que se utiliza es el siguiente:

```
sem_t *sem_open(char *name, int flag);
```

Un semáforo con nombre posee un nombre, un dueño y derechos de acceso similares a los de un archivo. El nombre de un semáforo es una cadena de caracteres que sigue la convención de nombrado de un archivo. La función `sem_open` establece una conexión entre un semáforo con nombre y una variable de tipo semáforo.

El primer argumento, `name`, es el nombre del semáforo. El segundo argumento determina si la función `sem_open` accede a un semáforo previamente creado o crea uno nuevo. Si `flag` es 0 indica que se va a utilizar un semáforo ya existente, en este caso no es necesario los dos últimos argumentos y se utiliza el segundo prototipo de `sem_open`. Si `flag` es `O_CREAT`, entonces se crea un semáforo nuevo y sí son necesarios los dos últimos argumentos, la modalidad que se debe utilizar de `sem_open` es la primera. El tercer argumento especifica los permisos del semáforo que se va a crear, de la misma forma que se especifica en la llamada al sistema `open` (ej: `600` indica permiso de lectura y escritura para el dueño). El cuarto argumento, `val`, indica el valor inicial del semáforo.

Devuelve la dirección de la variable que identifica el semáforo y -1 en caso de error.

Ejemplo de creación de un semáforo binario llamado "`semaforo`":

```
sem_t *semID=NULL;
sem= sem_open("semaforo",O_CREAT,0600,1);
```

**Cierre de un semáforo con nombre**

Cerrar un semáforo significa romper la asociación que se había creado entre un proceso y un semáforo al realizar `sem_open`. Se utiliza por parte de un proceso para indicar que ya ha terminado de utilizar el semáforo. Al cerrar el semáforo se elimina cualquier recurso del sistema utilizado por este proceso para el uso de este semáforo. El prototipo es el siguiente:

```
int sem_close(sem_t *sem);
```

El parámetro, `sem`, es el identificador del semáforo devuelto en la llamada a `sem_open`.

Devuelve 0 si éxito y –1 en caso de error.

Ejemplo del cierre de un semáforo cuyo identificador es `semID`:

```
sem_close(semID);
```

**Borrado de un semáforo con nombre**

El borrado de un semáforo con nombre significa la eliminación de este semáforo del sistema. La destrucción del semáforo se pospone hasta que todos los procesos que lo estén utilizando lo hayan cerrado con la función `sem_close`. El prototipo es el siguiente:

```
int sem_unlink(char *name);
```

El parámetro indica el nombre del semáforo.

Devuelve 0 en caso de éxito y –1 en caso de error.

Ejemplo de eliminación de un semáforo cuyo nombre es "semaforo":

```
sem_unlink("semaforo");
```

**Operación *wait***

La operación `wait` (también conocida como bajar) se consigue con el siguiente servicio:

```
int sem_wait(sem_t *sem);
```

El parámetro es el identificador del semáforo sobre el que se quiere realizar la operación `wait`.

Devuelve 0 en caso de éxito y –1 en caso de error.

Ejemplo de operación bajar sobre un semáforo cuyo identificador es `semID`:

```
sem_wait(semID);
```

**Operación *signal***

La operación `signal` (también conocida como subir) se consigue con el siguiente servicio:

```
int sem_post(sem_t *sem);
```

El parámetro es el identificador del semáforo sobre el que se quiere realizar la operación `signal`.

Devuelve 0 en caso de éxito y –1 en caso de error.

Ejemplo de operación subir sobre un semáforo cuyo identificador es `semID`:

```
sem_post(semID);
```

NOTA: El fichero de cabecera necesario para utilizar estos servicios es `semaphore.h`.

**TAREAS:**

1. Realizar el programa `crea_sem.c` que cree un semáforo nombrado. Como parámetros del programa se dará el nombre del semáforo y también el valor de inicio del semáforo. Ejemplo:

```
crea_sem semBinario 1
```

NOTA: para la compilación hay que indicar la librería adecuada (`-pthread`).

2. Realizar el programa `elimina_sem.c` que destruye un semáforo nombrado. Como parámetro del programa se debe indicar el semáforo que se quiere eliminar. Ejemplo:

```
elimina_sem semBinario
```

3. Realizar el programa `usa_sem.c` que utiliza un semáforo nombrado creado anteriormente con crea. El nombre del semáforo se da como parámetro al programa. Se va a considerar que la región crítica cuyo acceso exclusivo se garantiza mediante la utilización del semáforo es un bucle que imprime 5 (`NUM_IMP`) veces en pantalla. Para que el resultado sea visible, después de cada impresión de `HOLA` en pantalla se debe esperar 1s (`sleep(1)`). El programa debe ejecutar 5 (`VECES_RC`) veces esta región crítica esperando 1s de una ejecución a otra. Lo que se debe imprimir en pantalla es "RC: i IMP: j \n" donde i (de 1 a 5) es el número de la región crítica y j (de 1 a 5) el número de la impresión.

4. Crear un semáforo binario con `crea_sem`, usar el semáforo desde 2 o más procesos (probar primero con 2, luego con 3 y con 4) con `usa_sem`, razonar el resultado de la ejecución y una vez que terminen los procesos que usan el semáforo, destruirlo con `elimina_sem`.

5. Probar lo mismo que en el apartado anterior, pero iniciando el semáforo a 2.

6. Ejecutar la siguiente orden en el shell para ver que el semáforo se ha creado:

   ```
   > ls –l /dev/shm
   ```

   Si /dev/shm aparece enlazado a /run/shm entonces hay que dar:

   - `ls –l /run/shm`

7. Envía en la tarea habilitada para la práctica 5 los ficheros de los apartados anteriores `crea_sem.c, elimina_sem.c y usa_sem.c`.