Prácticas de Sistemas Operativos

Juan Antonio Álvarez, Toñi Reina, David Ruiz,
Antonio Tallón, Pablo Neira, José Ángel Bernal y Sergio Segura

Boletín ♯6: Semáforos

Curso 2006/07

Índice

---

1. Introducción

Para comunicar procesos en UNIX System V se dispone, entre otros, de tres mecanismos:

- los semáforos, que van a permitir sincronizar procesos, y que serán objeto de estudio en este boletín;
- la memoria compartida, para que los procesos puedan compartir su espacio de direcciones virtuales, que queda fuera del ámbito de estas clases de laboratorio.
- las colas de mensajes, que permiten el intercambio de datos con un formato determinado, y que serán la parte central del siguiente boletín.

Estos mecanismos se han implementado de manera uniforme y tienen una serie de características comunes, entre las que están:

- Cada mecanismo tiene una tabla cuyas entradas contienen información acerca del uso del mismo.
- Cada entrada de la tabla tiene una llave numérica elegida por el usuario.
- Cada mecanismo cuenta con una llamada get() para crear una entrada nueva o recuperar una existente. La llamada contendrá, el menos, dos argumentos: una llave, y una máscara de indicadores.

1.1. Llaves

Una llave es una variable o constante del tipo key_t que se utiliza para identificar colas de mensajes, memoria compartida, y grupos de semáforos.

Un programa puede obtener una llave de tres formas distintas:

- usando la constante IPC_PRIVATE, con lo que hacemos que el sistema genere una llave;
- escogiendo al azar un número llave;
- utilizando ftok() como veremos a continuación.

Para crear una llave System V se apoya en el concepto de proyecto, con la idea de que todos los mecanismos de comunicación entre procesos que pertenezcan a un mismo proyecto compartan el mismo identificador. Esto impide que procesos que no estén relacionados, por no formar parte de un mismo proyecto, puedan interferirse involuntariamente debido a que usan la misma llave.

La librería estándar de C de System V proporciona la función ftok() para convertir una ruta y un identificador de proyecto a una llave de IPC (*Interprocess Communication*, comunicación entre procesos). Esta función tiene la siguiente declaración:

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

key_t ftok(char *pathname, int id);
```

En el fichero de cabecera <sys/types.h> se define el tipo key_t, que suele ser un entero de 32 bits.

La función ftok devuelve una llave basada en pathname e id. pathname es un nombre de archivo que debe existir en el sistema, y al cual debe poder acceder el proceso que llama a ftok. id es un entero que identifica al proyecto. ftok devolverá la misma llave para rutas enlazadas a un mismo fichero, siempre que se utilice el mismo valor para id. Para el mismo fichero, ftok devolverá diferentes llaves para distintos valores de id.

Si el fichero no es accesible para el proceso, bien porque no existe, bien porque sus permisos lo impiden,ftok devolverá el valor -1, que indica que se ha producido un error en la creación de la llave.

Ejemplo 1 *El siguiente trozo de código muestra el uso de la función ftok:*

```
key_t llave;
...
   llave = ftok("prueba", 1);
   if (llave == (key_t) -1) {
       /* Error al crear la llave. Tratamiento del error */
   }
```

1.2. Facilidades de IPC desde la línea de comandos

Los programas estándar ipcs e ipcrm nos permiten controlar los recursos IPC que gestiona el sistema, y nos pueden ser de gran ayuda a la hora de depurar programas que utilizan estos mecanismos. ipcs se utiliza para ver qué mecanismos están asignados, y a quién. ipcrm se utiliza para liberar un mecanismo asignado. A continuación se explican las opciones más comunes de estos comandos:

ipcs: Si no se muestra ninguna opción, *ipcs* muestra un resumen de la información de control que se almacena para los semáforos, memoria compartida y mensajes que hay asignados. Las principales opciones son:

- **-q** Muestra información de las colas de mensajes que hay activas.
- **-m** Muestra información de los segmentos de memoria compartida que hay activos.
- **-s** Muestra información de los semáforos que hay activos.
- **-b** Muestra información completa sobre los tipos de mecanismos IPC que hay activos.

Ejemplo 2 *A continuación se muestra un ejemplo de la salida que mostraría el comando* ipcs

```
$ ipcs

IPC status from <running system> as of Mon May 5 13:50:44 MEST
2003
 T         ID      KEY        MODE        OWNER    GROUP
 Message Queues:
 q          0 0x3c08204d --rw-r--r-- root root
 Shared Memory:
 m 0 0x500020c7 --rw-r--r-- root root
 Semaphores:
 s 0 0x0109522 --rw-r--r-- root     root
```

ipcrm: Algunas de las opciones más comunes son:

- **-q msqid** Borra la cola de mensajes cuyo identificador coincide con msqid
- **-m shmid** Borra la zona de memoria compartida cuyo identificador coincide con shmid.
- **-s semid** Borra el semáforo cuyo identificador coincide con semid.

2. Semáforos en UNIX System V

El mecanismo de semáforos implementado en UNIX System V es una generalización del concepto de semáforo visto en teoría, ya que permite manejar un conjunto o grupo de semáforos con un identificador asociado. Cuando realicemos una operación Down o Up, éstas actuarán de forma atómica sobre los semáforos del grupos.

2.1. Creación/Acceso de semáforos. semget

Con la llamada semget podemos crear o acceder a un grupo de semáforos que tienen un identificador común. Además inicia todos los semáforos del grupo a 0:

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

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

donde:

key Es la llave que indica a qué grupo de semáforos queremos acceder. Se podrá obtener de una de las tres formas vistas en la introducción.

nsems Es el número total de semáforos que forman el grupo devuelto por semget. Cada uno de los elementos dentro del grupo de semáforos puede ser referenciado por los números enteros desde 0 hasta nsems-1.

semflg Es una máscara de bits que indica en qué modo se crea el semáforo, siendo:

IPC_CREAT Si este flag está activo, se creará el conjunto de semáforos, en caso de que no hayan sido ya creados.

IPC_EXCL Esta bandera se utiliza en conjunción con IPC_CREAT, para lograr que semget de un error en el caso de que se intente crear un grupo de semáforos que ya exista. En este caso, semget devolvería -1 e inicializaría la variable errno con un valor EEXIST.

permisos del semáforo Los 9 bits menos significativos de semflg indican los permisos del semáforo. Sus posibles valores son:

0400 Permiso de lectura para el usuario.

0200 Permiso de modificación para el usuario.

0060 Permiso de lectura y modificación para el grupo.

0006 Permiso de lectura y modificación para el resto de usuarios.

Para indicar los permisos se pueden utilizar las siguientes constantes definidas en el archivo de cabecera <sys/stat.h>:

```
#define S_IRUSR         00400   /* read permission: owner */
#define S_IWUSR         00200   /* write permission: owner */
#define S_IRGRP         00040   /* read permission: group */
#define S_IWGRP         00020   /* write permission: group */
#define S_IROTH         00004   /* read permission: other */
#define S_IWOTH         00002   /* write permission: other */
```

Si la llamada a semget funciona correctamente, devolverá un identificador del grupo de semáforos con el que podremos acceder a los semáforos en sucesivas llamadas. Si hay algún tipo de error, semget devuelve el valor -1 e introducirá en la variable global errno el código del error que se ha producido.

Ejemplo 3 *El siguiente código crea un grupo con dos semáforos a partir de un nombre de ruta.*

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

#define PERMISOS S_IRUSR | S_IWUSR
#define NUM_SEM 2

int sem;
key_t llave;

   llave = ftok ("/tmp/misemaf", 1);
   if ( llave != (key_t) -1)
```

```
{
  sem = semget (llave, NUM_SEM, PERMISOS | IPC_CREAT);
  if (sem ==  -1)
    /* Error al crear un nuevo grupo de semáforos con llave */
}
```

En este ejemplo, para asegurarnos de que creamos un nuevo grupo de semáforos, debemos levantar la bandera IPC_CREAT. Lo mismo ocurriría en el caso de utilizar una llave ya hecha.

2.2. Operaciones Down y Up sobre semáforos. semop

Para realizar las operaciones Down y Up vistas en teoría tenemos que utilizar la llamada semop. Su declaración es la siguiente:

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

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

semid Identificador del grupo de semáforos sobre el que se van a realizar las operaciones atómicas.

sops Es un puntero a un array de estructuras que indican las operaciones que se van a realizar sobre los semáforos.

nsops Es el número total de elementos que tiene el array de operaciones.

Cada elemento del array sops es una estructura de tipo sembuf que se define de la siguiente manera:

```
struct sembuf {
        short           sem_num;        /* semaphore # */
        short           sem_op;         /* semaphore operation */
        short           sem_flg;        /* operation flags */
};
```

donde

sem_num Es el número del semáforo. Su valor está en el rango 0.. N-1, siendo N el número total de semáforos que hay agrupados en el identificador. Este campo se utiliza como índice para acceder a un semáforo concreto del grupo.

sem_op Es la operación a realizar sobre el semáforo sem_num. El resultado es sumar el valor de sem_op al contador del semáforo, de forma que si:

- Si sem_op es positivo equivale a una operación Up.
- Si sem_op es negativo equivale a una operación Down.

sem_flg Son una serie de banderas para indicar cómo responderá la llamada sem_op en el caso de encontrarse con alguna dificultad.

semop devuelve 0 en caso de éxito y -1 en caso de error, conteniendo la variable errno el código correspondiente. Esto puede ocurrir, con una operación Down. Por ejemplo, si el semáforo tiene valor 2, no se le puede restar un número mayor que 2, porque el semáforo entonces pasaría a tener un valor negativo. Cuando no se puede ejecutar una operación, la respuesta de semop dependerá del valor del campo sem_flg. Estos pueden ser:

IPC_NOWAIT En este caso, semop devuelve el control si no se puede satisfacer la operación que se intenta hacer. Por defecto se trabaja con IPC_WAIT.

SEM_UNDO La operación se deshace cuando el proceso termina.

Si semop es interrumpido por una señal, entonces devuelve -1 y le da a la variable errno el valor EINTR. Una buena implementación de los semáforos debería reiniciar la operación solicitada en el caso de que fuera interrumpida por una señal, aunque este aspecto no lo tendremos en cuenta en las prácticas.

Ejemplo 4 *El siguiente trozo de código muestra cómo realizar una operación Down sobre el semáforo 3 de un grupo de cuatro semáforos, y otra operación Up sobre el semáforo 1.*

```
struct sembuf op_down_3 = {3, -1, 0};
struct sembuf op_up_1 = {1, 1, 0};

semop(semid, &op_down_3, 1);
semop(semid, &op_up_1, 1);
```

Como puede apreciarse en el código anterior, la asignación de valores es una operación muy repetitiva, por lo que se podría crear una función de la siguiente manera:

```
void Down(int semid, int nsem)
{
  struct sembuf op_down = {0, -1, 0};
  op_down.sem_num=nsem;
  result = semop(semid, &op_down, 1);
  if (result != 0)
   /* error en operación down */
}
```

Utilizando esta función, el código anterior quedará como sigue:

```
/* Operación Down para el semáforo de índice 3 */
Down(semid, 3);
```

2.3. Control de semáforos. semctl

La llamada semctl sirve para controlar un grupo de semáforos. Su declaración es la siguiente:

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

int semctl (int semid, int semnum, int cmd, /* union semun arg */... );

union semun{
    int val;
    struct semid_ds *buf;
    ushort *array;
    }arg;
```

donde:

semid Es el identificador del grupo de semáforos sobre el que actuará semctl. Previamente se ha tenido que crear con semget.

semnum Indica cuál es el semáforo del grupo de semáforos al que se quiere acceder. Su valor estará en el rango 0..N-1, siendo N el número de semáforos que hay en el grupo.

cmd Es la operación de control a realizar sobre el semáforo. Sus valores vienen definidos por las siguientes constantes:

GETVAL Se utiliza para leer el valor de un semáforo, que se devolverá como valor de retorno de semctl.

SETVAL Inicializa un semáforo a un valor determinado. Este valor se indica en el argumento arg.

GETPID Se utiliza para leer el PID del último proceso que actuó sobre el semáforo y se devuelve como valor de retorno de semctl.

GETNCNT Se utiliza para leer el número de procesos que hay esperando a que se incremente el valor del semáforo. Este valor se devuelve como valor de retorno de semctl.

GETZCNT Obtiene el número de procesos que están esperando a que el semáforo tome el valor 0, el cual, se devuelve como valor de retorno de semctl.

GETALL Permite leer el valor de todos los semáforos asociados al identificador semd. Estos valores se devuelven en el campo array de arg.

SETALL Se utiliza para inicializar el valor de todos los semáforos asociados al identificador semd. Los valores de inicialización deben estar en arg.

IPC_STAT Permite leer la información de control asociada al identificador semd. Para más información, consultar el manual de UNIX.

IPC_SET Permite modificar la información de control asociada al identificador semd. Para más información, consultar el manual de UNIX.

IPC_RMID Le indica al núcleo que tiene que borrar el conjunto de semáforos asociados al identificador semd. Esta operación no tendrá efecto mientras haya un proceso que esté usando los semáforos.

semctl devuelve un número cuyo significado dependerá del valor de cmd, de forma que, cuando éste sea GETVAL, el valor devuelto será el valor de un semáforo, o cuando sea GETNCNT será el número de procesos que esperan a que se incremente el valor de un semáforo. Así, tienen un significado especial el valor de retorno en el caso de que cmd valga GETVAL, GETNCNT, GETZCNT o GETPID. Para el resto de valores de cmd, semctl devuelve un cero. En el caso de que haya algún tipo de error, se devuelve un -1.

Ejemplo 5 *El siguiente trozo de código crea una función para borrar un conjunto de semáforos dado por un identificador mediante la función semctl. En ese caso no es necesario incluir el argumento opcional.*

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

int borrar_semaforo(int semd)
{
    return (semctl (semd, 0, IPC_RMID));
}
```

Ejemplo 6 *El siguiente trozo de código muestra una función que inicializa un semaforo concreto de un grupo de semáforos a un valor mediante la llamada semctl.*

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

int inicializar_semaforo(int semd, int numsem, int valor)
{
    union semun arg;    /* Debe definirse previamente la union de tipo semun*/

    arg.val = valor;
    return (semctl (semd, numsem, SETVAL, arg));
}
```

3. Ejercicios

1. Realice una biblioteca de funciones para simular los semáforos vistos en teoría haciendo uso de semáforos UNIX, es decir, grupos de un sólo semáforo sobre los que sólo se pueden realizar operaciones `Down` y `Up`. Para ello debe crear el siguiente tipo de datos:

   ```
   typedef int Semaph;
   ```

   este tipo identifica a un grupo de un único semáforo. Además, se deben proporcionar las siguientes funciones:

   - `Semaph Semaph_Create (char nombre_sem[], int valor_ini);`
     Esta función es la más importante de todas, ya que sirve crear un semáforo con nombre nombre_sem y asignarle un valor inicial valor_ini. Para ello se darán los siguientes pasos:
     a) Comprobar si existe un archivo llamado /tmp/semaph.uid.nombre_sem, donde uid es el identificador de usuario que crea el semáforo.
     b) En caso negativo, se debe generar una llave a partir de ese archivo y crear un grupo de un único semáforo. Además, debe iniciarse el valor del mismo a valor_ini.
     c) En caso afirmativo, sólo se tendrá que devolver una referencia al semáforo, es decir, devolver el resultado de semget.
   - `void Semaph_Destroy (char nombre_sem[], Semaph sem);`
     Esta función elimina el archivo /tmp/semaph.uid.nombre_sem y el grupo de un único semáforo sem.
   - `void Semaph_Down (Semaph sem);`
     Esta función realiza una operación Down sobre sem.
   - `void Semaph_Up (Semaph sem);`
     Esta función realiza una operación Up sobre sem.
   - `void Semaph_Set (Semaph sem, int val);`
     Esta función asignará el valor dado por val al semáforo.
   - `int Semaph_Get (Semaph sem);`
     Esta función devolverá el valor que tiene actualmente el semáforo.
   - `void Semaph_Print (Semaph sem);`
     Esta función imprimirá por la salida estándar de error el valor actual del semáforo.

   Para probar esta biblioteca puede utilizar el siguiente programa, que sincroniza la escritura de dos procesos por la salida estándar:

   ```
   #include <stdio.h>
   #include "Semaph.h"

   int main(void)
   {
     int i, pid;

     Semaph sem_p, sem_h;

     sem_p = Semaph_Create("padre",1);
     sem_h = Semaph_Create("hijo",0);

     if( sem_p == -1 || sem_h == -1)
        printf("Error al crear los semáforos") ;
     else
       {
         pid = fork();
         if (pid == -1 )
          {
   ```

```
            perror("Error de creación de proceso");
        }
        else if (pid == 0) /* hijo */
        {
            for(i=0; i < 5; i++)
             {
               Semaph_Down(sem_h);
               printf ("Soy el hijo %d \n",i);
               Semaph_Up(sem_p);
             }
            Semaph_Destroy("padre",sem_h);
        else /* padre */
        {
          for (i = 0;i < 5; i++)
            {
              Semaph_Down(sem_p);
              printf ("Soy el padre:  %d \n",i);
              Semaph_Up(sem_h);
              sleep(1);
            }
          Semaph_Destroy("hijo",sem_p);
        }
    }
}
```

que debe generar la siguiente salida:

```
~>./test
Creando padre sobre /tmp/semaph.230.padre
Asignando a padre valor 1
Creando hijo sobre /tmp/semaph.230.hijo
Asignando a hijo valor 0
Soy el padre:  0
Soy el hijo 0
Soy el padre:  1
Soy el hijo 1
Soy el padre:  2
Soy el hijo 2
Soy el padre:  3
Soy el hijo 3
Soy el padre:  4
Soy el hijo 4
Semaforo padre destruido
Semaforo hijo destruido
```

2. Se desea sincronizar, haciendo uso de semáforos, un grupo de procesos emisores con un proceso receptor (no existe relación filial entre dichos procesos). Para ello se utilizará un archivo `BUZON` (defínase como una constante) en el que los procesos emisores escribirán los mensajes introducidos por la entrada estándar y del que el proceso receptor leerá y escribirá en la salida estándar. Se tienen que tener en cuenta los siguientes aspectos de sincronización:

   a) Dos emisores no pueden escribir a la vez en el buzón.

   b) El receptor sólo ejecutará una lectura cuando exista algún mensaje disponible, después de leerlo se entenderá que no quedan mensajes disponibles.

   NOTA: Debe capturar la señal `SIGINT` para destruir los semáforos utilizados así como el archivo de buzón.

   Por ejemplo, si en un terminal tenemos:

```
> ./emisor
Proceso emisor.Mensaje: hola
Proceso emisor.Mensaje: estas sola?
Proceso emisor.Mensaje: ^C
Destruyendo semaforos...
Destruyendo buzon...
>
```

En el terminal en el que se ejecute el receptor se tendrá:

```
> ./receptor
Proceso receptor.Mensaje: hola
Proceso receptor.Mensaje: estas sola?
>
```

3. Se desea sincronizar a un grupo de procesos *'lectores'* con un proceso *'escritor'*, teniendo en cuenta las siguientes restricciones:

   - Varios lectores pueden ejecutar una lectura (simuladas con un `printf()`) más un retraso máximo de `T_LECTURA` segundos) siempre que el escritor no esté realizando una escritura (simulada de la misma forma con un tiempo de retraso máximo de `T_ESCRITURA` segundos).
   - El escritor tiene que esperar que todos los lectores terminen sus lecturas pendientes antes de comenzar una escritura.
   - El número de lectores existentes se almacenan en un archivo `NLECTORES` sobre el que se tiene que garantizar la exclusión mutua en su acceso (tanto de lectura como de escritura).

   NOTA: Tal y como se ha visto en teoría, en este escenario es posible que el escritor sufra inanición si los lectores nunca paran de leer. Para resolver este problema vamos a asumir que los lectores están `T_OCIOSO` segundos esperando entre lectura y lectura y el escritor `T_OCIOSO` segundos entre dos escrituras consecutivas.

A. Ejercicios de Examen

1. ***(Ejercicio de Examen 1ª CONV II 2002-03)***

   Escriba el código C de una función *int Crea_Dos_Semaforos(void)* que cree un grupo de dos semáforos con permisos *S_IRUSR | S_IWUSR*. En caso de error esta función devolverá -1, en caso de éxito devuelve un entero que identifica al grupo de semáforos. Si lo estima conveniente puede hacer uso de la llamada al sistema `mktemp` que sirve para generar un nombre de archivo único a partir de una plantilla. Su prototipo es el siguiente:

   ```
   char* mktemp(char * template)
   ```

   dónde template es una cadena de la forma prefijoXXXXXX.

   Por ejemplo, el siguiente programa:

   ```
   int main(void){ char tmpfilename[MAX];

     sprintf(tmpfilename,"semaph-XXXXXX");
     mktemp(tmpfilename);
     printf("%s\n", tmpfilename);
   }
   ```

   escribe en la salida estándar una cadena formada por el prefijo `semaph-` más seis caracteres de forma que se asegura que no existe ningún archivo en el directorio `/tmp` con ese nombre.

2. ***(Ejercicio de Examen 2ª CONV II 2002-03)***

   Se desea desea implementar una biblioteca de semáforos contadores para un sistema operativo que SÓLO dispone de un compilador de C y de semáforos binarios, que sólo pueden adoptar los valores cero y uno, con el siguiente interfaz:

   ```
   BSemaph BSemaph_Create(char nombre[], int valor_inicial);
   void    BSemaph_Destroy(char nombre[], BSemaph bs);
   void    BSemaph_Down(BSemaph bs);
   void    BSemaph_Up(Bsemap bs);
   ```

   Se pide:

   Apartado a .- Declarar los campos necesarios para el nuevo tipo de datos llamado `Semaph`. ¿Cuántos semáforos binarios necesita y por qué?

   Apartado b .- Escribir el código C que implemente una función constructora que dado un `nombre` y un `valor`, cree un nuevo semáforo contador con dicho nombre y dicho valor inicial.

   Apartado c .- Escribir dos funciones C para implementar las operaciones `Up` y `Down`.

3. ***(Ejercicio de Examen 1ª CONV ITI 2004-05)***

   Un grupo de amigos, simulados por procesos, se reúne semanalmente para jugar un partido de futbito. El número de jugadores debe ser de 10 para jugar 5 contra 5 pero hay ms de 10 amigos y a veces hay gente que se queda de suplente por si alguno falla. Por ese motivo hay dos listas, una de titulares (`TITULARES`) y otra de suplentes (`SUPLENTES`). Los 10 primeros en apuntarse a la lista serán titulares, mientras que los restantes irán a la lista de suplentes. Cuando se ha apuntado el décimo jugador en `TITULARES`, se generan 2 equipos mediante el proceso sorteo. Dicho proceso solo se ejecuta cuando la lista de titulares llega a 10 jugadores. Debido a problemas de diversa naturaleza, los jugadores apuntados a cualquier lista, pueden querer borrarse del partido. En ese caso si están en la lista de SUPLENTES, se borrarán de la misma y no habrá más modificaciones, en caso de estar en la de `TITULARES`, al borrarse, el primer suplente pasará a la lista de `TITULARES`, y el proceso sorteo volverá a realizar la asignación de equipos.

   Los procesos jugadores recibirán un argumento desde la línea de comandos que indica el nombre del amigo que quiere jugar. Si dicho jugador ya está apuntado significa que quiere borrarse. Contemple por tanto las dos posibilidades en el proceso jugador.

   Resumiendo, las condiciones de sincronización son las siguientes:

- El acceso a las listas `TITULARES` y `SUPLENTES`, se debe hacer de forma exclusiva.
- El proceso sorteo sólo se ejecutará cuando haya 10 entradas en la lista `TITULARES`.

Implemente, usando la librería de semáforos vista en clase y las funciones que mostramos a continuación, el proceso `jugador` y el proceso `sorteo`.

```
int busca (lista l, char *nombre);
```
Devuelve la posición en la lista l que ocupa el jugador con nombre igual al segundo parámetro. En caso de no estar en dicha lista devuelve -1.

```
void extrae(lista l, int indice, char *nombre);
```
Borra el elemento de la lista l que está en la posición índice y obtiene su nombre en el tercer argumento: nombre (parámetro de salida). Hace que los elementos de la lista se desplacen sin dejar ningún hueco libre.

```
void inserta(lista l, char *nombre);
```
Inserta el nombre de un jugador en la lista l.

```
int tamanyo (lista l);
```
Devuelve el tamaño de la lista.

```
void generaEquipos(lista l);
```
A partir de una lista se generan 2 equipos.

NOTAS: Suponga las listas `TITULARES` y `SUPLENTES` creadas. Además, no es necesario contemplar la finalización de procesos mediante señales.

Ejemplo de ejecución:

```
>Jugador pepe          >Jugador fernando          Proceso Sorteo activado.
Titular 1              Titular 7                  Equipos generados.
>Jugador antonio       >Jugador antonio           >Jugador francisco
Titular2               Titular 2 Borrado. Lista   Suplente 1
>Jugador manolo        desplazada. 6 titulares    >Jugador joaquin
Titular3               >Jugador daniel            Titular 9 Borrado Lista
>Jugador alejandro     Titular 7                  desplazada. francisco titular.
Titular 4              >Jugador alvaro        Proceso Sorteo activado.
>Jugador jose          Titular 8                  Equipos generados.
Titular 5              >Jugador joaquin
>Jugador pablo         Titular 9
Titular 6              >Jugador javier
                       Titular 10
```