Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

PRÁCTICA 6. MEMORIA COMPARTIDA

1. OBJETIVO

El objetivo de esta práctica es comprender las llamadas al sistema para la utilización de memoria compartida. El acceso a esta memoria compartida se gestiona mediante la utilización de semáforos.

2. DESCRIPCIÓN

Los procesos que se ejecutan en una misma máquina tienen espacios de direcciones diferentes, por lo tanto, el espacio de memoria utilizado por un proceso no puede ser accedido por ningún otro proceso. A veces es útil que dos o más procesos puedan compartir información utilizando una zona de memoria común.

Los servicios correspondientes a memoria compartida permiten crear un área de memoria que será compartida por más de un proceso. Esta área puede ser creada por un proceso y posteriormente escrita y leída por cualquier número de procesos.

Para la utilización de una zona de memoria compartida se tienen que llevar a cabo los siguientes pasos:

- Creación de un nuevo segmento de memoria compartida o acceder a uno existente.

  Llamada al sistema: **`shmget`**

- Mapeado del segmento de memoria compartida al espacio de direcciones del proceso.

  Llamada al sistema: **`shmat`**

- Lectura o escritura. En esta zona se lee y se escribe como en cualquier dirección de memoria del proceso. Por ser una zona de memoria compartida, para realizar el acceso a esta zona hay que utilizar semáforos para obtener acceso exclusivo.

  Llamadas al sistema correspondientes a semáforos ya vistas en prácticas anteriores.

- Desenlace del segmento de memoria compartida por parte del proceso.

  Llamada al sistema: **`shmdt`**

Claves IPC

Antes de comenzar a estudiar las llamadas al sistema para la utilización de memoria compartida es necesario explicar que son las claves IPC (InterProcess Communication). Cada objeto IPC tiene un identificador único, que se usa dentro del núcleo para identificar de forma única un objeto IPC. Una zona de memoria compartida es un objeto IPC, ya que es un objeto que sirve para la comunicación entre procesos (otros objetos IPC son los mensajes o los semáforos).

Para obtener el identificador único correspondiente a un objeto IPC se debe utilizar una *clave*. Esta clave debe ser conocida por los procesos que utilizan el objeto. La clave puede ser estática, incluyendo su código en la propia aplicación. Pero en este caso hay que tener cuidado ya que la clave podría estar en uso. Otra forma es generar la clave utilizando la función `ftok` cuyo prototipo es el siguiente:

```
key_t ftok(char *nombre, char proj);
```

`ftok` devuelve una clave basada en `nombre` y `proj` que puede ser utilizada para la obtención de un identificador único de objeto IPC. El argumento `nombre` debe ser el nombre de camino de un fichero existente. Esta función devolverá la misma clave para todos los caminos que nombren el mismo fichero, cuando se llama con el mismo valor para `proj`. Devolverá valores diferentes para la clave cuando es llamada con caminos que nombren a distintos ficheros o con valores diferentes para el parámetro `proj`.

Ejemplo:

```
key_t clave;
clave=ftok(".",'S');
```

Creación de un segmento de memoria compartida

Para la creación de un nuevo segmento de memoria compartida o para acceder a uno existente se utiliza el servicio `shmget`. Permite obtener el identificador del objeto de memoria compartida a partir de una clave dada. El prototipo es el siguiente:

```
int shmget(key_t key, int size, int shmflg);
```

Los parámetros son los siguientes:
`key`: clave asociada al objeto de memoria compartida que se quiere crear o acceder.
`size`: tamaño del área de memoria compartida.
`shmflg`: flag indicando los permisos de acceso y algunas condiciones

- `IPC_CREAT`: crea un segmento si no existe ya en el núcleo.
- `IPC_EXCL`: al usarlo con `IPC_CREAT`, falla si el segmento ya existe.

Devuelve el identificador del segmento de memoria compartida si éxito o –1 en caso de error.

Ejemplos:

Para obtener el identificador de un segmento compartido creándolo si no existe:

```
if ((shmid= shmget(clave,100,IPC_CREAT|0660))!=-1)
{
/* shmid es el identificador de memoria compartida */
}
```

Para crear un segmento de memoria compartida en caso de que el segmento no exista, pero no hacer nada en el caso de que exista:

```
if ((shmid= shmget(clave,100,IPC_CREAT|IPC_EXCL|0660))!=-1)
{
/* shmid es el identificador de memoria compartida */
}
else
{
/* el segmento no se ha creado */
}
```

Mapeado del segmento de memoria compartida

Una vez que se obtiene el identificador de la zona de memoria compartida, se debe mapear al espacio de direcciones del proceso para poder utilizar esta zona de la misma forma que cualquier dirección de memoria. Para ello se utiliza el servicio shmat. El prototipo es el siguiente:

```
char *shmat(int shmid, char *shmaddr, int shmflg);
```

Los parámetros son los siguientes:

`shmid`: identificador del área de memoria compartida
`shmaddr`: dirección donde se mapea el área de memoria compartida. Si es NULL, el núcleo intenta encontrar una zona no mapeada.
`shmflg`: se indica el flag SHM_RDONLY si es solo para lectura.

Devuelve la dirección de memoria donde se ha mapeado el área de memoria compartida o –1 en caso de error.

Ejemplo:

```
char *addr;
addr=shmat(shmid,NULL,0);
```

Desenlace del segmento de memoria compartida por parte del proceso

Una vez que ya no se necesita más acceder al segmento de memoria compartida, el proceso debe realizar el desenlace. El desenlace no es lo mismo que la eliminación del segmento desde el

núcleo. El segmento de memoria compartida se elimina sólo en el caso de que no queden procesos enlazados. Para el desenlace se utiliza el servicio `shmdt`. El prototipo es el siguiente:

```
int shmdt(char *shmaddr);
```

Donde `shmaddr` es la dirección de memoria del segmento devuelta por `shmat`.

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

Manipulación del segmento de memoria compartida

El prototipo para realizar operaciones de control sobre un segmento de memoria compartida existente es el siguiente:

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

Donde `shmid` es el identificador de la zona de memoria compartida, `cmd` es la operación que se quiere realizar y `buf` es una estructura donde se guarda la información asociada al segmento de memoria compartida en caso de que la operación sea `IPC_STAT` o `IPC_SET`. Las operaciones que se pueden realizar son:

`IPC_STAT`: guarda la información asociada al segmento de memoria compartida en la estructura apuntada por buf. Esta información es por ejemplo el tamaño del segmento, el identificador del proceso que lo ha creado, los permisos, etc.

IPC_SET: Establece los permisos del segmento de memoria a los de la estructura `buf`.

IPC_RMID: Marca el segmento para borrado. No se borra hasta que no haya ningún proceso que esté asociado a él.

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

Ejemplo:

```
shmctl(shmid,IPC_RMID,NULL);
```

Marca el segmento de memoria compartida cuyo identificador es shmid para borrado.

NOTA: también se puede acceder a memoria compartida con las llamadas shm_open y shm_unlink. Pero para ello es necesario conocer las llamadas al sistema para el mapeo de ficheros que aún no se han visto, pero que se verán en una práctica posterior.

⇒ **Ejemplo:**

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <string.h>
#include <ctype.h>
```

```
#define TAM_SEG 100

int main(int argc, char **argv)
{
  key_t clave;
  int shmid;
  char *seg;

  clave=ftok(".",'S');

  if((shmid=shmget(clave,TAM_SEG,IPC_CREAT|IPC_EXCL|0660))==-1)
      {
        printf("El segmento de memoria compartida ya existe\n");
        printf("    Abriendo como cliente\n");

        if((shmid=shmget(clave,TAM_SEG,0))==-1)
            printf("Error al abrir el segmento\n");
      }
  else
      printf("Nuevo segmento creado\n");

  if(argc==1)
      printf("Error de argumentos\n");

  else if (shmid != -1)
      {
        if((seg=shmat(shmid,NULL,0))== (char *)-1)
            printf("Error al mapear el segmento\n");
        else
            switch(tolower(argv[1][0]))
              {
              case 'e': /* escribir en el segmento */
                  if(argc==3)
                    {
                        strncpy(seg,argv[2],TAM_SEG-1);
                        if(strlen(argv[2])>=TAM_SEG-1)
                          seg[TAM_SEG-1]='\0';
                        printf("Hecho...\n");
                    }
                  break;

              case 'l': /* Leer del segmento */
                  printf("El contenido del segmento es: %s\n", seg);
                  break;

              case 'b':
                  shmctl(shmid,IPC_RMID,0);
                  printf("Segmento de memoria marcado para borrar\n");
                  break;
              default:
                  break;
              }

        shmdt(seg);

      }

  return(0);

}
```

P6_prueba1.c

TAREAS:

1. Descarga el fichero p6_prueba1.c correspondiente a la práctica 6.
2. Compila y ejecuta el programa p6_prueba1.c. Estudia el ejemplo.
3. Realizar el problema del productor/consumidor utilizando una zona de memoria compartida entre los procesos. Para ello:
   - En el fichero creaMC.c crear e inicializar los semáforos necesarios y el segmento de memoria compartida. Los semáforos que se usan son huecos (número de huecos en el buffer, iniciado al tamaño del buffer), elementos (número de elementos en el buffer, iniciado a 0) y mutex (para acceso exclusivo a la región crítica, iniciado a 1). También debe inicializar las variables de entrada y salida del buffer que indican la posición de entrada en el buffer que usarán los productores cuando vayan a incluir un número en el buffer y la posición de salida del buffer que usarán los consumidores cuando vayan a extraer un elemento del buffer. El tamaño del buffer va a ser de 10 (TAM_BUFFER) elementos de tipo entero. En la memoria compartida se debe tener el buffer y además dos enteros adicionales para las variables de entrada y salida que se pueden poner a continuación del buffer.
   - En el fichero eliminaMC.c destruir los semáforos y el segmento de memoria compartida.
   - En el fichero productorMC.c poner el productor que debe producir 30 elementos. Los pasos en el productor son los siguientes:

     Abrir semáforos
     Obtener id de la MC y mapear

     Para cada dato a incluir en el buffer
       Imprimir dato
       Bajar huecos
       Bajar mutex
       Poner dato en el buffer en el lugar adecuado
       Subir mutex
       Subir elementos

     Desproyectar la MC
     Cerrar semáforos

   - En el fichero consumidorMC.c poner el consumidor que debe consumir 15 elementos. Los pasos en el consumidor son los siguientes:

     Abrir semáforos
     Obtener id de la MC y mapear

     Para cada dato a recuperar del buffer
       Bajar elementos
       Bajar mutex
       Tomar dato del buffer del lugar adecuado
       Subir mutex

Esperar 1 seg para poder ver la salida
Subir huecos
Imprimir dato

Desproyectar la MC
Cerrar semáforos

NOTA: El buffer se trata como un buffer circular, es decir, que las variables ent y sal se van incrementando, pero al llegar al final del buffer vuelven a 0. Esto se hace usando el módulo de la siguiente forma: ent = ent+1 % TAM_BUFFER.
Se puede ver el segmento de memoria compartida usando
> ipcs
Se puede borrar el segmento de memoria compartida usando
> ipcrm -M clave
> ipcrm -m id

4. Ejecutar primero el productor y luego el consumidor. Ejecutar primero el consumidor y luego el productor. Probar a tener un productor y varios consumidores. Probar a tener varios productores y un consumidor. Probar varios productores y varios consumidores.
5. Envía en la tarea habilitada para la práctica 6 los ficheros creaMC.c, eliminaMC.c, productorMC.c y consumidorMC.c del apartado 3.