Programación Avanzada

Curso 2023/2024

prof. Claudio Rossi

Practica 4: memoria compartida

La memoria compartida es el método más rápido para intercambiar datos entre procesos. Al utilizar un área especifica de memoria de forma compartida, el dato es accesible directamente a ambos procesos sin tener que solicitar gestión al sistema operativo para el transporte de los datos. Para evitar conflictos, habrá que usar mecanismos de sincronización de los procesos (p.e. "busy waiting, como en esta práctica, o semáforos, próxima práctica).
Nota: un segmento de memoria compartida es un bloque de bytes sin formato. Para comodidad de uso, se suele mapear ("*cast*") en estructuras de datos definidas comúnmente entre los procesos.

Objetivos:

1. Uso básico de memoria compartidas
2. Problema propuesto

Comandos de S.O.:

- `ipcs`: imprime en pantalla información del estado de las primitivas de IPC (colas, semáforos, memoria compartida)
- `ipcrm`: elimina estructuras de IPC (*ejemplos:* `ipcrm -m 123456` *elimina la zona de memoria compartida con id. 123456*)

Parte 1 (memoria compartida)

Estucturas de datos utilizadas:

- *struct datos,* presenta tres campos: *int elEntero, float elFloat* y una cadena de caracteres *char elArray[10]*:

```
typedef struct {
    int unEntero;
    float unFloat;
    char unArray[10];
}  datos;
```

Funciones utilizadas:

- int *shmget(int key, int size, int flag)*: proporciona el identificador de segmento de la zona de memoria compartida asociado a *key*. Si la zona de memoria compartida existe, devuelve su

identificador, en caso contrario la crea, aplicando la información contenida en *flag* (máscara de bits con instrucciones para la creación y permisos de acceso con formato rwxrwxrwx)

- char **shmat(int shmid, char *shmaddr, int flag)*: proporciona un puntero a la estructura de datos de la zona de memoria compartida. El primer parámetro es el identificador de segmento obtenido a través de *shmget*(), el segundo se refiere a la dirección de mapeo (0 = la dirección es elegida por el S.0, si no el S.O. intenta asignar la memoria en la dirección especificada), mientras que el tercer paámetro es un *flag* que indica modos de funcionamiento (p.e. SHM_RDONLY=solo lectura).

- int *shmctl(int shmid, int cmd, struct shmid_ds *buff)*: proporciona varios modos de control sobre la zona de memoria compartida con identificador *shmid*. Los modos quedan definidos por *cmd*, por ejemplo:
  - IPC_RMID, borra la memoria
  - SHM_LOCK: bloquea la zona de memoria (previene *swapping*)
  - SHM_UNLOCK: desbloquea la zona de memoria (permite *swapping*)
- El tercer parámetro, *buff*, es un puntero a una estructura con campos para almacenar el usuario, grupo de usuarios, etc. (no usados aquí).

- void *perror*(const char *message): imprime un mensaje de error asociado a la cadena que tiene como argumento.

Se escribirán dos programas. El primero crea una zona de memoria compartida de mensajes (*shmget*), y escribe información en ella, usando la estructura pactada entre los procesos (definida en el fuchero "common.h"). El segundo se conecta a la memoria creada (*shmget*), lee su contenido, mapeandolo en la estructura datos para poder usar correctamente su la información leída, y finalmente borra la memoria (*shmctl*). Para compartir la memoria, los dos programas deberán compartir la misma "clave" *key*.

Comprobad, a través del comando del S.O. *ipcs*, la creación del segmento compartido!

Fichero "common.h"

```
/* Datos comunes */

/* Clave de acceso a la zona de memoria compartida */
#define CLAVE_SHM ((key_t) 1001)

/* Estructura de datos que se comparte en la zona de memoria común */
typedef struct{
  int unEntero;
  float unFloat;
  char unArray[10];
}  datos;
```

Programa 1

```
/* Creación de una zona de memoria compartida */
```

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

/* Fichero que contiene la información común (clave y estructura' de datos) */
#include "common.h"


int main(void)
{
  int shmid;
  datos *zona_comun;

  shmid = shmget(CLAVE_SHM, sizeof(datos), IPC_CREAT|0666);

  if(shmid== -1)
      {
        perror("shmget");
        exit(EXIT_FAILURE);
      }

  zona_comun = (datos*) shmat(shmid,0,0);

  zona_comun->unEntero = 123;
  zona_comun->unFloat  = 1.23;
  strcpy(zona_comun->unArray,"hola mundo");

  exit(EXIT_SUCCESS);

}
```

Programa 2

```
/* Creación de una zona de memoria compartida */

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


/* Fichero que contiene la información común (clave y estructura de datos) */
#include "common.h"


int main(void)
{
  int shmid;
  datos *zona_comun;
  int res;

  shmid = shmget(CLAVE_SHM, sizeof(datos), IPC_CREAT|0666);
```

```
  if(shmid== -1)
      {
        perror("shmget");
        exit(EXIT_FAILURE);
      }

  zona_comun = (datos*) shmat(shmid,0,0);

  printf("El contenido de la memoria común es:  %d  %g  %s\n",
             zona_comun->unEntero,
             zona_comun->unFloat,
             zona_comun->unArray );


  /* Borrado de la zona de memoria compartida */
  shmctl(shmid, IPC_RMID,0);

  if(res==-1)
      {
        perror("shmctl");
        exit(EXIT_FAILURE);
      }

  exit(EXIT_SUCCESS);

}
```

Parte 2 (problema propuesto)

Basándose en los programas de ejemplo proporcionados, y en las practicas anteriores, escribir otros dos para que se cree una zona de memoria compartida para comunicación entre dos procesos. El proceso 1 deberá adivinar un numero elegido por el proceso 2 (entre 0 y 100). Para ello, escribirá sus intentos a través de la zona de memoria compartida. El proceso 2 contestará escribiendo en la zona de memoria compartida si el numero es mayor, menor o si ha acertado. En caso de acierto los proceso terminan. Para alternar los accesos, su usará la técnica de la espera activa ("busy waiting"). Para ello, se usará una variable "turno" en la memoria compartida, que se usará de la siguiente forma:

Proceso 1

```
while("no he acertado")
{
   while( zona_comun->turno==2) {/*busy waiting*/}
   // es mi turno
   // accedo a la memoria (leo respuesta y actualizo intento)
    // cedo el turno:     zona_comun->turno=2;
}
```

Proceso 2

```
while( zona_comun->turno==1) {/*busy waiting*/}
   // es mi turno
   // accedo a la memoria (leo intento y actualizo respuesta)

   // cedo el turno:    zona_comun->turno=1
```