Programación Avanzada (MUAR)

Curso 2023/2024

prof. Claudio Rossi

Practica 5: sincronización de proceso a través de semaforos

Los semáforos representan una primitiva del S.O. para coordinar o sincronizar actividades en las cuales múltiples procesos compiten por los mismos recurso, como por ejemplo un fichero, una zona de memoria compartida etc. Un semáforo es una variable entera gestionada por el sistema operativo al que cada proceso puede acceder para leer y escribir. Típicamente, un proceso que utiliza semáforos chequea el valor, y entonces, si utiliza el recurso, cambia el semáforo para reflejar la reserva del recurso de forma de que cualquier otro proceso que pida el recurso esperará a que se libere.
Las operaciones de *signal* y *wait* (también llamadas *up* y *down)*, consisten en efecto en comprobar el valor del semáforo, y seguir si es mayor a cero, disminuyendo su valor. Si el valor del semáforo es cero, el proceso deberá bloquearse, siendo puesto en *sleep* (espera), y deberá esperar un tiempo hasta intentar otra vez.

Objetivos:
1. Uso básico de semáforos
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 (*ejemplo:* `ipcrm -s 123456` *elimina el semáforo con id. 123456*)

Estucturas de datos utilizadas:

- *struct sembuf,* presenta tres campos:

```
typedef struct sembuf{
short sem_num; /* Semáforo sobre el que actuar   */
short sem_op;  /* Operación a realizar +1, -1    */
short sem_flg; /* Opciones: IPC_NOWAIT, SEM_UNDO */
}
```

Funciones utilizadas:

- int *semget(int key, int nsems, int flag)*: proporciona el identificador para un conjunto de *nsems* semáforos, asociados a la llave *key*. Si el conjunto de semáforos ya existe proporciona su identificador y en caso contrario los crea con los permisos indicados en *flag* (máscara de

bits con instrucciones para la creación y permisos de acceso con formato rwxrwxrwx)

- int *semctl(int semid, int semnum, int cmd, union semun arg)*: permite la realización de distintas operaciones de control sobre el conjunto de semáforos agrupados en *semid*. La variable *semnum* establece el semáforo seleccionado, mientras que *cmd* indica la operación ha efectuar. Por ejemplo
  - SETVAL pone un valor al semaforo
  - IPC_RMID borra el semaforo.

  La union *arg* presenta distintos campos donde se puede almacenar el valor de los semáforos, estados de los procesos, etc.

- int *semop(int semid, struct sembuf *sops, unsigned nsops)*: permite realizar operaciones sobre los semáforos asociados al identificador *semid*. *sops* es un array de estructuras *sembuf* (ver arriba) que contienen comandos, *nsops* es el numero de dichas estructuras.

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

Parte 1.
Se escribirán dos programas. El primero crea un semáforo (*semget*), e inicializa su valor a 1. El segundo se conecta al semáforo creado (*semget*). Los dos intentarán acceder a un recurso (simulado), haciendo una operación de *wait* antes de entrar en su sección critica, y una *signal* al dejarla. Finalmente, tras 10 accesos, el proceso 1 borra el semaforo (*semctl*). Para compartir el semaforo, los dos programas deberán compartir la misma "clave" *key*, definida en el fichero "common.h".

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

Fichero "common.h"

```
/* Datos comunes */


/* Definición de los nombres de los semáforos  */
#define SEM_1 0
#define SEM_2 1
#define CLAVE 1

// función que implementa una pausa de duracción casual entre 1 y 3 segundos
void pausa()
{
  int pausa;

  pausa=1000+(int)(2000*(random()/(float)RAND_MAX));   // entre 1000 y 3000
microsegundos
  usleep(pausa*1000);  // entre 1000 y 3000 millisegundos
}
```

Programa 1

```
/* Creación de un semaforo y ciclos de espera */

#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdlib.h>
#include <unistd.h>

#include "common.h"


int main(void)
{
  int i;
  int semid;
  struct sembuf operacion;

  // inicializa la semilla de los numeros casuales
  srandom(getpid());

  // creacción del semaforo
  semid = semget(CLAVE,1,IPC_CREAT|0600);

  if(semid == -1)
      {
        perror("semget");
        exit(EXIT_FAILURE);
      }

  /* Inicialización del semáforo */
  semctl(semid,SEM_1,SETVAL,1);

  printf("Valor semáforo: %d\n", semctl(semid,SEM_1,GETVAL,0));

  operacion.sem_flg=0;  /* No se activa ninguna opcion */

  i=10;

  while(i>0)
      {
        i--;

        printf("A: espero sem1 (WAIT)\n");
        operacion.sem_num = SEM_1;  // que semaforo (puede haber más de 1!)
        operacion.sem_op = -1;      // que operación: +1=signal, -1=wait
        semop(semid,&operacion,1);  // envía el comando

        printf("Proceso A dentro ! (%d)\n",i);

        pausa(); // simula la actividad en la region critica

        printf("A: libero sem1 (SIGNAL)\n");
        operacion.sem_num = SEM_1;
        operacion.sem_op = 1; // que operación: +1=signal, -1=wait
        semop(semid,&operacion,1);
        printf("Proceso A fuera ! (%d)\n",i);
```

```
        pausa(); // simula la actividad fuera de la region critica

      }

  /* Elimina el semáforo */
  semctl(semid,0,IPC_RMID,0);

  printf("Fin !\n");
}
```

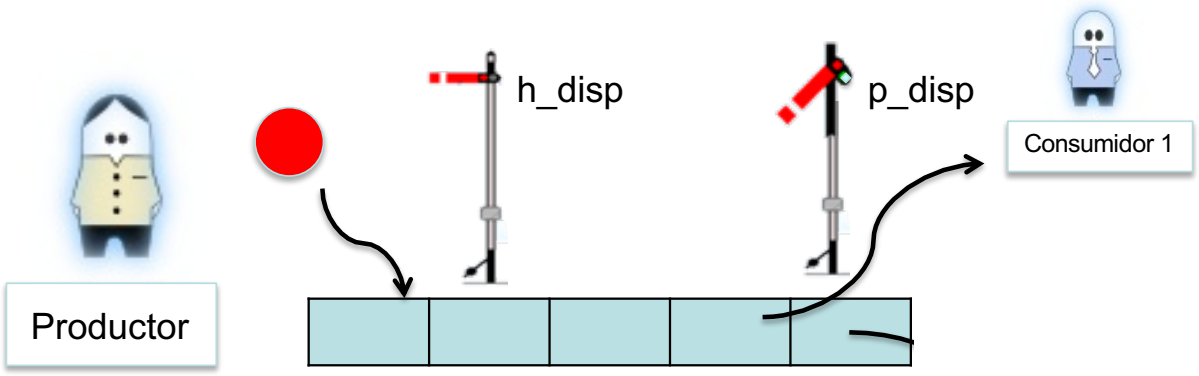
Programa 2

```
/* Uso de semaforos, programa 2 */
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <stdlib.h>
#include <unistd.h>

#include "common.h"


int main(void)
{
  int i;
  int semid;
  struct sembuf operacion;

  // inicializa la semilla de los numeros casuales
  srandom(getpid());

  // creacción del semaforo
  semid = semget(CLAVE,1,IPC_CREAT|0600);

  if(semid == -1)
      {
        perror("semget");
        exit(EXIT_FAILURE);
      }

  // el proceso 2 NO inicializa el semaforo !
  printf("Valor semáforo: %d\n", semctl(semid,SEM_1,GETVAL,0));

  operacion.sem_flg=0;  /* No activamos ninguna opcion */

  i=10;

  while(i>0)
      {
        i--;

        printf("B: espero sem1\n");
        operacion.sem_num = SEM_1;
        operacion.sem_op = -1;
        semop(semid,&operacion,1);
```

```
        printf("Proceso B dentro ! (%d)\n",i);

        pausa(); // simula la actividad en la region critica

        printf("B: libero sem1\n");
        operacion.sem_num = SEM_1;
        operacion.sem_op = 1;
        semop(semid,&operacion,1);
        printf("Proceso B fuera ! (%d)\n",i);

        pausa(); // simula la actividad fuera de la region critica

      }

  /* Elimina el semáforo */
  semctl(semid,0,IPC_RMID,0);

  printf("Fin !\n");
}
```

Parte 2.

Basándose en los programas de ejemplo proporcionados, y en la practicas anteriores, escribir dos programas que implementen una variante del problema del productor/consumidor (conocido también como "bounded-buffer problem"). En este problema, hay dos proceso que comparten un buffer de tamaño limitado. Uno de ellos, el productor, usa el buffer para colocar unos *ítems,* y el segundo (el cosumidor) coge estos *ítems* del buffer. El productor puede colocar un *ítems* en el buffer solo si hay un hueco libre, si no tiene que esperar hasta que haya (al menos) uno. El consumidor recoge un *ítems* en el buffer solo si hay (al menos) uno, si no tiene que esperar hasta que haya uno. Para implementar correctamente el funcionamiento de este sistema, se usarán dos semáforos, uno para señalar cuando hay elementos disponibles en el buffer y uno para señalar cuando hay huecos libres en el buffer.



Adicionalmente, se usará un tercer semáforo para mutua exclusión para evitar que los dos procesos accedan al buffer a la vez.