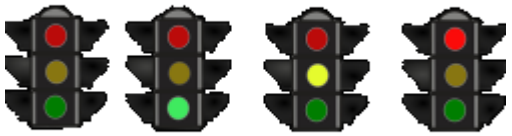
Sincronización Padre-Hijos con Semáforos

C Prácticas, UNIDAD 3 - Sistema de Procesos, Unidad 3 Tema 5



En esta entrada vamos a ver un ejemplo de cómo sincronizar padre-hijos con semáforos utilizando lo expuesto en la entrada Semáforos Se parte del ejemplo Comunicación Padre-Hijos mediante señales, pero ahora quitamos las señales y ponemos semáforos. Definiremos una barrera para parar al padre y un semáforo que sincroniza a cada hijo.

El enunciado del problema es similar al del ejercicio anterior:

Tenemos un proceso padre que tiene que esperar a que un conjunto de procesos hijos, por ejemplo 5, lleguen a un determinado punto del código. Implementar una barrera para este hecho. Una vez que el proceso padre detecta que todos los hijos han pasado la barrera, entonces comienza a enviar ACKs a todos los hijos. Implementar la solución utilizando semáforos.

Aquí no vamos a tener señales, por lo que todos los manejadores y demás código relacionado con señales va fuera.

Vamos a utilizar semáforos, necesitamos varios:

- Un semáforo que actuará de barrera entre hijos y padre. Lo llamaremos sBarrera.
- Un semáforo para cada hijo donde se colgará a esperar el ACK, es decir, un array de semáforos, uno para cada hijo, que los colocaremos en un array llamado sACKS.

Los hijos harán signal sobre el semáforo sBarrera y el padre hará tantos wait como hijos hayan sobre el mismo semáforo. Esto implementa la barrera.

Una vez el hijo hace signal sobre el sBarrera, hace un wait sobre su semáforo de ACK. El padre cuando pasa la barrera hace un signal al semáforo de ACK de cada hijo, para que este continúe.

Vamos a hacer uso de la librería sem.h / sem.c definida en la entrada Semáforos. Aunque esto nos obliga a infrautilizar las capacidades de definición de semáforos que tiene linux, como explicaré más adelante.

Definiremos un proyecto formado por los siguientes ficheros:

- sem.h Fichero de cabecera de la librería definida en Semáforos.
- sem.c Fichero fuente de la librería definida en Semáforos
- semaforos.c Fichero fuente con el ejemplo de este artículo
- makefile Fichero makefile para compilar con make

Makefile

Un posible makefile para el proyecto sería:

```
CC=gcc
CFLAGS=-I.
INC=sem.h
OBJECTS=semaforos.o sem.o

semaforos: $(OBJECTS)
 $(CC) $(OBJECTS) $(CFLAGS) -o semaforos
```

```
semaforos.o: semaforos.c $(INC)
 $(CC) -c semaforos.c  $(CFLAGS)

sem.o: sem.c $(INC)
 $(CC) -c sem.c  $(CFLAGS)

clean:
 rm semaforos $(OBJECTS)
```

sem.h

Este es el mismo código definido en su momento en Semáforos

```
//Creamos un semáforo con un valor inicial
int sCreate(int seed, int value);

//Operaciones de incremento y decremento del semáforo
void sWait(int semID);
void sSignal(int semID);
```

sem.c

Este es el mismo código definido en su momento en Semáforos

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

#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#define PERMISOS 0644

typedef union semun {
    int val;               /* used for SETVAL only */
    struct semid_ds *buf; /* for IPC_STAT and IPC_SET */
    ushort *array;         /* used for GETALL and SETALL */
    } semun;


int sGet(int seed)
{
```

```
 key_t semKey = ftok("/bin/ls", seed);
    if (semKey == -1 ) { perror("Error al crear el token: "); exit(-1);}
 int semID = semget(semKey,1,IPC_CREAT | PERMISOS);
    if (semID == -1 ) { perror("Error al obtener el semID: "); exit(-1);}
 return semID;
}

void sSet(int semID, int value)
{
 semctl(semID, 0, SETVAL, value);
}

int sCreate(int seed, int value)
{
 int semID = sGet(seed);
 sSet(semID, value);
 return semID;
}

void sWait(int semID)
{
 struct sembuf op_Wait [] =  { 0, -1, 0 }; // Decrementa en 1 el semáforo
 semop ( semID, op_Wait, 1 );
}

void sSignal(int semID)
{
 struct sembuf op_Signal [] = { 0, 1, 0 }; // Incrementa en 1 el semáforo
 semop ( semID, op_Signal, 1 );
}

void sDestroy(int semID)
{
    semun dummy;
    int ret;

    ret=semctl(semID, 0, IPC_RMID, dummy);
    if (ret == -1 ) { perror("Error al obtener el semID: "); exit(-1);}
}
```

semaforos.c

Este sería un ejemplo que implementa el enunciado usando las funciones de sem.h

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

```c
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include "./sem.h"

#define NUMHIJOS 5
#define SSEED 99

//fijamos el número de hijos
const int numHijos = NUMHIJOS;

int main( int args, char *argv[] )
{
 int i;
 int pid;
 struct sigaction action;
 int status;
 char buffer[100];

 //Declaramos el identificador del semáforo que juega el rol de barrera del padre
 int sBarreraPadre;

 //Creamos el semáforo inicializado a cero
 sBarreraPadre=sCreate(SSEED,0);

 //Declararmos el array de semáforos para los ACKs de los hijos y los inicializamos a cero
 int sACKS[NUMHIJOS];
 for(i=0; i<numHijos; i++)
 sACKS[i]=sCreate(SSEED+i+1,0);

 //---***Bucle para crear a los hijos***---
 //En el bucle los hijos envían su señal, reciben su ACK y terminan.
 for(i=0; i < numHijos; i++) { //El padre itera creando hijos
 pid=fork();
 if(pid == -1){
    sprintf(buffer,"ERROR. Fork ha fallado al crear al hijo %d\n",i);
    perror(buffer);
    exit(-1);
 }
 if(pid == 0){
 //Código hijo
    printf("Hijo %d llega a su barrera.\n", getpid());

    //El hijo indica que ha llegado a la barrera
    sSignal(sBarreraPadre);

    printf("Hijo %d esperando ACK\n", getpid());
```

```
    //El hijo espera en su semaforo de ACK para seguir.
    sWait(sACKS[i]);

                    //El hijo destruye su semaforo
                    sDestroy(sACKS[i]);

    printf("El hijo %d termina.\n",getpid());
    exit(0); //El hijo termina, no itera.
  }
  }
    //Aquí sólo llega el padre

    //El padre espera a que todos los hijos lleguen a la barrera
  for (i=0; i<numHijos; i++){
  sWait(sBarreraPadre);
  printf("Un hijo ha llegado a su barrera\n");
  }
  printf("Todos los hijos han pasado su barrera\n");

  printf("El padre envía ACKs\n");
  //Enviamos los ACKS
  for (i=0; i<numHijos; i++){
  sSignal(sACKS[i]);
  }

  printf("El padre entra a esperar la terminación de los hijos\n");
  //Bucle para esperar la terminación de los hijos y notificarlo
  for(i=0; i < numHijos; i++){
  pid=waitpid(-1,&status,0); //Esperamos la terminación de cualquier hijo recogiendo el pid del mismo.
  printf("El hijo con pid %d ha terminado\n",pid);
  }

        //El padre destruye su semaforo
        sDestroy(sBarreraPadre);
  return 0;
}
```

Definimos un número entero como semilla para los semáforos. Al crear un semáforo se le pasa el entero. Este número sirve para crear un identificador de recurso compartido único para el semáforo. Utilizaremos este número para crear el semáforo sBarrera que es compartido por padre e hijos. Para crear un semáforo diferente para cada hijo tenemos que usar un número distinto de semilla para crear semáforos diferentes. Podríamos pensar que el pid del hijo podría valer como semilla, pero el pid lo conocemos tras el fork, con lo que el hijo no heredaría el semáforo. Al hacerlo antes del fork usaremos SSEED + valor.

Declaramos y creamos el array de sBarrera utilizando la función sCreate, pasándole SSEED como semilla y lo inicializamos a 0 (segundo parámetro). Así cuando alguien haga wait se colgará si está a cero.

Declaramos e inicializamos un array de semáforos sACKS[NUMHIJOS], es decir, un array de tantos enteros como hijos. En cada indice del array colocamos el identificador de un semáforo distinto creado en el bucle con sCreate(SSEED+i+1,0) inicializándolos también a cero.

En el bucle de creación de los hijos, en el código del hijo, cada hijo señaliza su barrera, es decir hace signal al llegar a su barrera usando la función sSignal(sBarrera). Ésto no lo bloquea, por lo que sigue su ejecución. Pero el hijo debe esperar un ACK del padre. Para parar al hijo hasta que lo reciba, hará un wait utilizando sWait sobre su semáforo que está en el índice apropiado del array, es decir sWait(sACKS[i]). Aquí esperará a que el padre envíe el ACK. Realmente el padre no envía nada y el hijo no recibe nada, simplemente se sincronizan en ese punto mediante un semáforo.

El padre, una vez creados los hijos, se pone a esperar en su barrera, es decir, hace tantos sWait(sBarrera) como hijos tiene, de forma que cada sSignal de un hijo lo saca del bloqueo. Al salir del bucle porque todos los hijos han llegado a su barrera, el padre se pone a enviar los ACKS, que hemos dicho que se trata simplemente de señalizar los semáforos de los hijos, por lo que hace sSignal sobre cada uno de los semáforos del array. Con esto, el hijo podrá continuar al recibir el signal, y termina.

Por último, como en el ejemplo anterior, el padre se pone a esperar la terminación de los hijos, para no dejar hijos huérfanos.

Como os comentaba anteriormente, esto de crear un array de enteros donde guardar un semáforo diferente para cada hijo, podríamos haberlo hecho utilizando un único identificador para un array de semáforos creados con la función semget, tal como se explica en Semáforos. Si queréis podéis ampliar la librería sem.c para ofrecer una función que simplifique la creación de un array de semáforos bajo un mismo identificador y por tanto habría que modificar u ofrecer nuevas funciones wait y signal que permitan seleccionar el indice del semáforo a usar cuando tenemos un identificador único.

La forma en la que lo hemos definido aquí es más parecida a los ejemplos que se han visto en clase cuando usamos arrays de semáforos.

← Entrada anterior

Entrada siguiente →

Escuela Politécnica Superior de Elche

Universidad Miguel Hernández

Miguel Onofre Martínez Rach