Sincronización Padre-Hijos con semáforos y memoria compartida

C Prácticas

En esta entrada vamos a ver un ejemplo básico de cómo sincronizar padre-hijos con semáforos Semáforos y compartiremos memoria compartida entre ellos con lo explicado en Memoria Compartida. Se modifica el ejemplo Sincronización Padre-Hijos con semáforos para añadirle memoria compartida.

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 escribe en una zona de memoria compartida que heredan los hijos un conjunto de valores enteros, luego comienza a enviar ACKs a todos los hijos.

Tras recibir el ACK los hijos leen y muestran el contenido de la memoria compartida. Implementar la solución utilizando semáforos y memoria compartida.

El ejemplo tiene el mismo código, en lo que respecta a semáforos, del ejemplo mostrado en Sincronización Padre-Hijos con semáforos. A dicho código le hemos añadido las librerías necesarias para tratar con memoria compartida y hemos utilizado las funciones descritas en Memoria Compartida.

El padre, antes de la creación de los hijos, crea y se vincula a un segmento de memoria compartida, que inicializa a cero. Una vez que los hijos son creados, estos heredan el puntero al segmento de memoria compartida obtenido en el padre, con lo que podrán leer/escribir en ella. Pero sincronizamos los procesos padre-hijos de forma que primero el padre espera a que todos los hijos lleguen a una barrera y se quedan colgados (wait) en un semáforo a la espera de que el padre haga signal sobre el mismo y puedan continuar. Cuando todos los hijos han llegado a su barrera, el padre escribe en la memoria compartida, modificando los valores iniciales. Libera entonces a los hijos para que estos puedan leer la memoria ya modificada. Una vez leida, los hijos se desvinculan del segmento de memoria compartida y terminan. El padre a su vez, tras la terminación de todos sus hijos se desvincula del segmento, lo elimina y termina.

El código fuente sería el siguiente, sólo muestro el código del fichero shared.c puesto que las librerías sem.c y sem.h son las mismas que en el ejemplo citado. Adjunto también el makefile para compilar el proyecto. Podéis juntar todos los ficheros en un directorio para probar.

shared.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include "./sem.h"

#define NUMHIJOS 5
#define SSEED 99
#define SHMSEED 35
#define SHMPERMISOS 0644

//fijamos el número de hijos
```

```c
const int numHijos = NUMHIJOS;

int main( int args, char *argv[] )
{
 int i,j;
 int pid;
 struct sigaction action;
 int status;
 char buffer[100];
 int shmId, shmSize;
 int *shmArray; //Puntero del proceso donde queremos ubicar el segmento de memoria compartida.

 //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);

 //Creamos el segmento de memoria compartida, Un array de 6 enteros.
 shmSize = 6 * sizeof(int);
 printf("Tamaño del segemento a pedir: %d\n",shmSize);

 //Creamos el segmento y asignamos correctamente los punteros.
 shmId = shmget(ftok("/bin/ls", SHMSEED), shmSize, IPC_CREAT | SHMPERMISOS);
 shmArray = shmat(shmId,0,0);

 printf("El padre inicializa el array de memoria compartida con id: %d\n",shmId);
 memset(shmArray,0,shmSize);

 //---***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 lee la memoria compartida
printf("Hijo %d lee Array Compartido con id %d\n",getpid(),shmId);
for (j=0; j<6; j++)
printf("%d ",shmArray[j]);
printf("\n");

//El hijo se desvincula del array.
printf("El hijo %d se desvincula del array\n", getpid());
shmdt(shmArray);

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");

//El padre escribe en la memoria compartida, vemos el valor inicial y el nuevo.
printf("El padre escribe el array de memoria compartida con id: %d\n",shmId);
for (i=0; i<6; i++){
printf("%d->",shmArray[i]);
shmArray[i]=i;
printf("%d ",shmArray[i]);
}
printf("\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 se desvincula del array.
 printf("El padre se desvincula del array\n");
 shmdt(shmArray);

 //El padre borra el array
 printf("El padre se destruye del array\n");
 shmctl(shmId,IPC_RMID,0);

 printf("El padre termina\n");
 return 0;
}
```

makefile

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

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

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

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

clean:
 rm shared $(OBJECTS)
```

← Entrada anterior

Entrada siguiente →