# Esqueleto Práctica Parte C – Semáforos

C Prácticas

## Versión con Semáforos

Ahora la espera del padre a que todos los hijos hayan apostado se va a realizar con un semáforo que juega el rol de barrera.

Podéis ver un ejemplo en Sincronización Padre-Hijos con semáforos. Aquí, a diferencia del ejemplo, como utilizamos un pipe donde los hijos se quedan esperando, no es necesario sincronizar con semáforos a los hijos que esperan la combinación del padre.

Respecto a la versión 2, simplemente tendremos que eliminar el código relativo a las señales, tanto del código de main.c como de funciones.h y funciones.c. También tendremos que añadir al proyecto la librería sem.c / sem.h que hemos definido en Semáforos.

Os dejo únicamente el esqueleto de los ficheros main.c, funciones.h y funciones.c, puesto que el código de sem.h y sem.c podéis obtenerlo de Semáforos o de Sincronización Padre-Hijos con semáforos.

Recordar que deberéis hacer un makefile para compilarlos.

## main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <sys/types.h>
#include "funciones.h"
#include "sem.h"

// DEFINES Y CONSTANTES GLOBALES



int main(int argc, char *argv[]){

//Declaración e inicialización de los semáforos usados para la sincronización,
//un semáforo que actuará como barrera compartida entre padre-hijos.


//Declaración e inicialización de otras variables globales

...


//Creamos el pipe o los pipes y controlamos que se haya creado correctamente.
//Determinar el protocolo a usar tanto si empleamos un único pipe para todos los hijos o un pipe por hijo


//Creamos las estrucutras hijos
HIJO hijos[numHijos]; // Declaramos el vector de estructuras HIJO

for(i=0; i<numHijos;i++){ //Inicializamos las estructuras de hijos
  hijos[i].pid=0;
  hijos[i].num=0;
  hijos[i].premio=0;
  //Añadimos el pipe correspondiente al hijo a la estructura si hemos decidido crear un pipe por hijo
}

//Creamos el bucle principal donde se llama a fork en cada iteración del padre.
```

```
//Los hijos hacen exit en su código de forma que no iteran.
for(i=0;i<numHijos;i++){
   //Código del padre
  //mantenimiento de la estructura de hijos

   //Código del hijo
  //Generamos la apuesta del hijo.

  //Señalizamos la barrera que compartimos con el padre

  //Leemos del Pipe, aquí se bloquea el hijo si todavía no está la combinación ganadora en el. (no es necesario recibir ACK con un semáforo de sincronización)

  //Calculamos los aciertos

  //Salimos comunicando aciertos
     exit(aciertos);
}
//... este código sólo lo ejecuta el padre tras todas las iteraciones...

//Esperamos a que todos los hijos apuesten

//Generamos la combinacion ganadora

//Escribimos la combinación ganadora en el/los pipes

//Esperamos a que todos los hijos lleguen a su barrera, lo que indica que han apostado

//Obenenemos el valor de retorno de cada hijo y calculamos el importe de su premio

//Generamos el fichero de resultados del sorteo en curso y salimos
}
```

# funciones.h

```
/*ESQUELETO DEL FICHERO func.c donde se define el código de las funciones a utilizar*/
#include <stdio.h>
#include <sys/types.h>

typedef struct {
    int pid ; // Pid del hijo
    int num ; // Numero de orden del hijo
    int num_aciertos; // Numero de aciertos.
 int pipehijo[2]; // Descriptores el pipe del hijo
    long premio; En mi caso el premio se escribe directamente en el fichero.
} HIJO ;
```

```c
int ValidarArgumentos(int argc, char** argv);
void CrearTuberias(int numHijos);
void ComunicarApuesta(int numHijo, unsigned int *ganadora);
void ApuestaRealizada();
int ComprobarCombinacion(unsigned int *ganadora, unsigned int *apuesta);
int CalcularPremio(int numAciertos);
void LeerGanadora(int numHijo, unsigned int *ganadora);
void GenerarCombinacion(unsigned int *combinacion);
```

# funciones.c

```c
/*ESQUELETO DEL FICHERO func.c donde se define el código de las funciones a utilizar*/
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <sys/time.h>
#include "funciones.h"

//FUNCIONES  ----------------------------------------------------
//FUNCIONES SOLO PADRE ////////////////////////////////////////////////////
int ValidarArgumentos(int argc, char** argv){
//Se podría utilizar esta función para validar los argumentos que recibe el programa sorteo.
//Los parámetros que recibe son los que recibe el propio main.
//El valor de retorno puede ser un código de OK u Error
}

void CrearTuberias(int numHijos){
//Esta función sirve para que el padre, antes de crear los hijos y sabiendo cuántos son
//cree todas las tuberias.
//Una forma es crear un array de tuberías global (que heredarán los hijos)

}

void ComunicarApuesta(int numHijo, unsigned int *ganadora){
//Esta función servirá al padre para enviar la combinación ganadora por la tubería correspondiente
//al hijo, siendo numHijo el índice del hijo en el array de tuberías.
}


//FUNCIONES SOLO HIJOS ////////////////////////////////////////////////////
void ApuestaRealizada(){
//Esta función la utilzarán los hijos para enviar una señal al padre de forma que éste
//sepa que el hijo ha realizado su apuesta.
}

int ComprobarCombinacion(unsigned int *ganadora, unsigned int *apuesta){
```

```
//Esta función la usarán los hijos para comprobar el número de aciertos de su combinación
//La función devuelve el número de aciertos que tendrá el jugador
}

int CalcularPremio(int numAciertos){
//Esta función servirá para calcular el importe del premio en función de la cantidad de números acertados
//Devuelve el premio en euros
}

void LeerGanadora(int numHijo, unsigned int *ganadora){
//Esta función servirá al hijo para leer la combinación ganadora de su tubería, que vendrá indicada
//por el número de hijo numHijo que es el índice que el hijo tiene en el array de tuberías.
//La combinación ganadora se depositará en el array ganadora.
}

//FUNCIONES COMUNES PADRE HIJOS ////////////////////////////////////////////////////
void GenerarCombinacion(unsigned int *combinacion){
//Esta función la utilizarán el padre y el hijo para generar la combinación
//El padre generará la combinación ganadora y el hijo su apuesta.
//Hay que inicializar el generador de números aleatorios con valores distintos para el padre y para cada hijo
//Lo ideal es que la combinación se devuelva ordenada o bien hacer uso de otra función para ordenarla como qsort
}
```

← Entrada anterior

Entrada siguiente →

## Deja un comentario

Conectado como alu28. Edita tu perfil. ¿Salir? Los campos obligatorios están marcados con *

Escribe aquí...

Publicar comentario »