# Esqueleto Práctica Parte C – Señales 1

C Prácticas

## Señales versión 1

Os dejo un posible esqueleto de cómo podríais programar la parte C de la primera versión de la práctica.

En esta versión utilizaremos señales (bidireccionalmente) para sincronizar a los procesos padres e hijos.

El hecho es que aunque los hijos no tendrían porqué esperar una señal como veremos en otras versiones, éstos esperan a que el padre haya generado la apuesta ganadora para ponerse a leer del pipe. Esta sincronización es mediante la espera a la llegada de la señal del padre que les permite continuar, es decir, generar su apuesta.

No tenéis porque seguir al pie de la letra el esqueleto, simplemente es una guía para que tengáis en cuenta cómo podéis resolverlo.

Tenéis tres ficheros que los llamáis como queráis, una idea sería

- funciones.h
- funciones.c
- main.c.

Os dejo a continuación el esqueleto de cada uno. Recordar que deberéis hacer un makefile para compilarlos.

## funciones.h

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

void HijoApuesta(int sig, siginfo_t *siginfo, void *context);
void SorteoRealizado (int sig, siginfo_t *siginfo, void *context);
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

```
/*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>
```

```c
#include "funciones.h"

//MANEJADORES ----------------------------------------------
//PADRE - MANEJADOR PARA LAS SEÑALES EN TIEMPO REAL ------
void HijoApuesta(int sig, siginfo_t *siginfo, void *context){
// Esta función es el manejador de señales en tiempo real.
// Cada hijo enviará al padre una señal diciendo que ha apostado
// Aquí contaremos cuántos hijos han contestado en una variable global que habrá que definir
}

//HIJOS - MANEJADOR DE SEÑALES NORMALES ------
void SorteoRealizado (int sig, siginfo_t *siginfo, void *context)
//Esta función es el manejador de la señal que reciben los hijos para enterarse que el padre
//ya ha generado la combinación ganadora, es decir, que se ha realizado el sorteo.
}

//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){
```

```c
//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
}
```

# main.c

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

// DEFINES Y CONSTANTES GLOBALES


int main(int argc, char *argv[]){
//Se define la estructura para el manejador de las señales que recibe el padre
struct sigaction manejador;

//Se inicializa el contador de mensajes recibidos, que han sido enviados por los hijos al haber realizado su apuesta
manejador.sa_sigaction =   ;//Nombre de la funcion manejadora
manejador.sa_flags = SA_SIGINFO; //Esto lo definimos para enviar informacion adicional junto a la señal
sigaction(SIGRTMIN, &manejador , NULL); //Establecemos el manejador de la señal a la función manejadora

//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.

  //Comunicamos al padre que hemos apostado

     //Espera el aviso del padre de que la combinación ganadora ha sido generada

  //Leemos del Pipe

  //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

//Informamos a los hijos que la combinación ya está generada

//Esperamos a la terminación de todos los hijos en un bucle con pause()
 //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
}
```

← Entrada anterior

Entrada siguiente →

## 6 comentarios en “Esqueleto Práctica Parte C – Señales 1”

4 DICIEMBRE, 2016 A LAS 21:45

Hola Miguel, en el código del hijo del main.c
//Espera el aviso del padre de que la combinación ganadora ha sido generada
//Leemos del Pipe

¿La parte de esperar el aviso del padre es necesaria?
En el momento lea del pipe quedará bloqueado hasta que el padre escriba en el mismo la combinación ganadora ¿estoy equivocado?

Gracias.

Responder

**MIGUEL ONOFRE MARTÍNEZ RACH**
5 DICIEMBRE, 2016 A LAS 10:37

Hola,
No, no es necesaria para generar un punto de sincronización entre hijos y padres.
Al leer del pipe, si este no tiene datos, el proceso se queda bloqueado hasta que lleguen los datos solicitados.
Pero en la primera versión os pido que probéis con una sincronización de los hijos mediante una señal del padre.
En la segunda versión ya quitamos esta señal del padre y procedemos como propones.
Revisar los efectos producidos en esta entrada.

Responder

2 DICIEMBRE, 2016 A LAS 21:18

int ValidarArgumentos(int argc, char** argv){
//Se podría utilizar esta función para validar los argumentos que recibe el programa sorteo.

¿es necesaria esta función?
¿no se encargaba el programa loteria de validar los argumentos?

Responder

**MIGUEL ONOFRE MARTÍNEZ RACH**
3 DICIEMBRE, 2016 A LAS 20:05

Hola,

En principio si sólo nosotros vamos a utilizar el programa y siempre lo vamos a lanzar desde el bash, no haría falta como dices.
Pero planteate que el programa sorteo en C puede ser llamado desde la línea de comandos, por ejemplo si queremos probarlo cuando no tengamos terminado la parte Bash, en ese caso necesitaríamos controlar los parámetros.
En cualquier caso siempre es bueno comprobar los parámetros en nuestros programas.

Responder

2 DICIEMBRE, 2016 A LAS 17:43

Buenas Miguel, ¿Por qué los Handler tanto para padre como para hijo son static?
“static void HijoApuesta(int sig, siginfo_t *siginfo, void *context);”

Responder

**MIGUEL ONOFRE MARTÍNEZ RACH**
3 DICIEMBRE, 2016 A LAS 20:01

Hola, debe haber sido un copy paste.
Una función static sólo es visible para las funciones dentro de su fichero de compilación.
Si los handler los ponemos en un fichero separado como es el caso, no sería necesario poner static.
Corregido.

En cualquier caso fíjate cómo podemos hacer un único handler para las dos señales. Puedes verlo en el Comunicación Padre-Hijos mediante señales

Responder

## Deja un comentario

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

Escribe aquí...

Publicar comentario »