7 Multithreading e interrupciones con Wiring Pi

7.1 Interupciones y multithreading con Wiring Pi[10]

WiringPi proporciona funciones auxiliares que permiten gestionar interrupciones externas, lanzar un nuevo hilo desde un programa, y administrar la prioridad de programa (o hilo/thread). Los subprocesos se ejecutan simultáneamente con el programa principal y se pueden utilizar para múltiples propósitos. Los hilos funcionan como los "Posix Threads".

7.2 Procesamiento concurrente (multi-threading)

WiringPi ofrece una interfaz simplificada para la implementación de los hilos Posix en Linux, así como un mecanismo (simplificado) para acceder a las exclusiones mutuas.

Usando estas funciones se puede crear un nuevo proceso (una función dentro del programa principal) que se ejecuta simultáneamente con el programa principal y pasarse entre ellos las variables de forma segura utilizando los mecanismos de exclusión mutua (mutex).

- **int piThreadCreate (nombre);**

Esta función crea un subproceso que ejecuta otra función en el programa que se ha declarado previamente mediante **PI_THREAD**. Esta función se ejecuta simultáneamente con su programa principal. Por ejemplo se puede hacer que esta función espere una interrupción mientras el programa continúa realizando otras tareas. El hilo puede indicar un evento o una acción mediante el uso de variables globales para comunicarse de nuevo con el programa principal u otros subprocesos.

Las funciones de hilo se declaran de la siguiente manera:

```
PI_THREAD (myThread)
{
.. Código que se ejecuta simultáneamente con el programa
    principal, probablemente en un bucle infinito}
}
```

y se inicia en el programa principal con

```
x = piThreadCreate (myThread);
if (x != 0) printf ("it didn't startn")
```

Esto no es más que una interfaz simplificada al mecanismo de subprocesos Posix que soporta Linux. Para saber más, se puede consultar el manual de los hilos Posix (man pthread).

```
piLock (int keyNum) ;
piUnlock (int keyNum) ;
```

Estas instrucciones permiten sincronizar las actualizaciones de las variables de su programa principal a cualquier subproceso que se ejecuta en su programa. KeyNum es

[10] Wiring Pi: GPIO Interface library for the Raspberry Pi http://wiringpi.com/reference/priority-interrupts-and-threads/

un número de 0 a 3 y representa una "clave". Cuando otro proceso intenta bloquear la misma clave, se paralizará hasta que el primer proceso haya desbloqueado esa clave.

Se necesita utilizar estas funciones para asegurarse de que se obtienen resultados válidos al intercambiar datos entre un programa principal y un hilo. De lo contrario, es posible que el hilo pueda despertar durante la copia de datos y cambiarlos de modo que la copia quede incompleta o no sea válida. Consultar el programa wfi.c en el directorio de ejemplos para ver un ejemplo.

7.3 Prioridad de programa o de hilo

int piHiPri (int prioridad);

Esta instrucción aumenta la prioridad del programa (o subproceso en un programa multihilo) y permite una planificación en tiempo real. El parámetro de **prioridad** va de 0 (predeterminado) a 99 (máximo). Esto no hace que el programa vaya más rápido, pero le da más tiempo cuando se están ejecutando con otros programas. El parámetro de prioridad es relativo a los otros, por lo que se puede establecer una prioridad de programa a 1 y otra prioridad a 2 y tendrá el mismo efecto que establecer una a 10 y la otra a 90 (siempre y cuando no se ejecuten otros programas con prioridades más altas).

El valor de retorno es 0 para éxito y -1 para error. Si se devuelve un error, el programa puede consultar la variable global errno, según las convenciones habituales.

Nota: Sólo los programas que se ejecutan como root pueden cambiar su prioridad. Si se llama desde un programa no root, no sucede nada.

7.4 Interrupciones

La versión más reciente del kernel (ampliada con el código de manejo de interrupciones GPIO) permite al programa esperar una interrupción. Esto libera el procesador para realizar otras tareas mientras espera. El GPIO puede configurarse para interrumpir por flanco de subida, de bajada o ambos flancos de la señal entrante.

- **int wiringPiISR (int pin, int edgeType, void (*function)(void)) ;**

Esta función apunta a la rutina/función de atención a la interrupción recibida en el pin especificado. El parámetro edgeType es **INT_EDGE_FALLING, INT_EDGE_RISING, INT_EDGE_BOTH, o INT_EDGE_SETUP**. Si se trata de **INT_EDGE_SETUP** no se producirá inicialización del pin -se supone que ya se ha configurado el pin en otro lugar (por ejemplo, con el programa gpio), pero si especifica uno de los otros tipos, el pin se exportará y se inicializará como se especifica (lo que se hace a través de una llamada adecuada al programa de utilidad gpio, que debe estar disponible).

El número de pin se puede suministrar en los modos wiringPi nativo, BCM_GPIO, físico o Sys.

Esta función funciona en cualquier modo y no necesita privilegios de root para ejecutarse.

La rutina/función se llamará cuando se active la interrupción. Cuando se dispara la interrupción, se borra del dispatcher antes de llamar a la función de atención, por lo que si una interrupción posterior se activa antes de terminar el procesamiento de la actual no se perderá. Sin embargo, sólo puede seguir una interrupción más. Si se dispara más de una interrupción mientras se está atendiendo a otra, entonces se ignorarán.

Esta función se ejecuta con prioridad alta (si el programa se ejecuta utilizando sudo o como root) y concurrentemente con el programa principal. Tiene acceso completo a todas las variables globales, manejadores de archivos abiertos, etc.

7.5 Ejemplos

7.5.1 Ejemplo de lectura de una entrada por encuesta

```
/* Programa encuesta.c
* Enciende un LED conectado al pin físico 7 (pin lógico
    Wiring Pi: 7) * y lee cíclicamente un pulsador
    (normalmente en alto) conectado al * * pin físico 5
    (pin lógico Wiring Pi: 9)
*********************************************************/
#include <stdio.h>
#include <wiringPi.h>
#include <softPwm.h>

#define LED_PIN 7
#define BUTTON_PIN 9

/*Programa principal*/
int main()
{
int pulsador=1;
wiringPiSetup();

pinMode(BUTTON_PIN,INPUT);
pinMode(LED_PIN,OUTPUT);

printf("El LED se apagará cuando se pulse el pulsador más de
    un segundo\n ");

      while (pulsador == 1)
      {
        digitalWrite(LED_PIN,HIGH);
        delay (800);
        digitalWrite(LED_PIN,LOW);
        delay (200);
        pulsador = digitalRead(BUTTON_PIN);
        printf("El estado del pulsador es: %d \n",
    pulsador);
      }

printf("Pulsador pulsado \n");
return 0;
}
```

7.5.2 Lectura de una entrada por interrupción

```
/* Programa interrupcion.c
* Enciende un LED conectado al pin físico 7 (pin lógico
    Wiring Pi: 7) * y espera a la interrupción generada por
    el pulsador (normalmente en * alto) conectado al pin
    físico 5 (pin lógico Wiring Pi: 9)
*************************************************************/

#include <stdio.h>
#include <wiringPi.h>
#include <softPwm.h>

#define LED_PIN 7
#define BUTTON_PIN 9
static volatile int fin=0;

/*Rutina de atención a la interrupción*/
void esperaInterrupcion(void)
    {
    fin=1;
    printf("Se ha pulsado el pulsador: %d \n", fin);
    }

/*Programa principal*/
int main()
{
int i, x, led;
wiringPiSetup();
pinMode(LED_PIN,OUTPUT);

/*Activa la espera por interrupción*/
wiringPiISR(BUTTON_PIN, INT_EDGE_BOTH, &esperaInterrupcion);

/*Enciende el LED mientras fin sea falso*/
printf("LED encendido. Esperando interrupción\n ");
while (fin==0)
      {
        digitalWrite(LED_PIN,HIGH);
        delay (900);
        digitalWrite(LED_PIN,LOW);
        delay (100);
      }

return 0;
}
```

7.5.3 Uso de threads

```
/*************************************************
*Programa test threads.c
*Crea dos threads que se ejecutan en paralelo con el main
*************************************************/
//inclusión librerias necesarias para el programa
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
//definición costantes
#define GLOBAL_UNIT_TIME 1000 // unidad de pausa temporal

//Código de cada thread
PI_THREAD (thread1)
{
  while(true)
  {
…
  }
}

PI_THREAD (thread2)
{
  while(true)
  {
…
  }
}

//Main program
int main(void)
{
//creación de los threads
  printf("Main: creando threads\n");
  int r,v;
  /*  @piThreadCreate: creatión de un thread. Param: nombre
    del thread; Return: 0 si no falla  */
    r = piThreadCreate(thread1);
    v = piThreadCreate(thread2);

  if((r != 0) || (v != 0)
  {
    printf("Error en la creación de un thread!\n");
    return 1;
  }

  for(;;)// Los programas de control no terminan nunca
  {
        …
  }
  return 0;
}
```