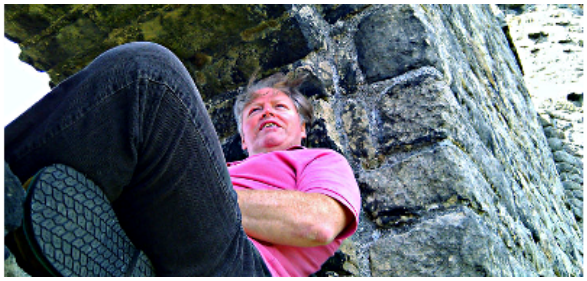
Wiring Pi

GPIO Interface library for the Raspberry Pi



Priority, Interrupts and Threads

WiringPi provides some helper functions to allow you to manage your program (or thread) priority and to help launch a new thread from inside your program. Threads run concurrently with your main program and can be used for a variety of purposes. To learn more about threads, search for “Posix Threads”

Program or Thread Priority

- **int piHiPri (int priority) ;**

  This attempts to shift your program (or thread in a multi-threaded program) to a higher priority and enables a real-time scheduling. The **priority** parameter should be from 0 (the default) to 99 (the maximum). This won’t make your program go any faster, but it will give it a bigger slice of time when other programs are running. The priority parameter works relative to others – so you can make one program priority 1 and another priority 2 and it will have the same effect as setting one to 10 and the other to 90 (as long as no other programs are running with elevated priorities)

  The return value is 0 for success and -1 for error. If an error is returned, the program should then consult the *errno* global variable, as per the usual conventions.

  Note: Only programs running as root can change their priority. If called from a non-root program then nothing happens.

Interrupts

With a newer kernel patched with the GPIO interrupt handling code, you can now wait for an interrupt in your program. This frees up the processor to do other tasks while you’re waiting for that interrupt. The GPIO can be set to interrupt on a rising, falling or both edges of the incoming signal.

Note: Jan 2013: The waitForInterrupt() function is deprecated – you should use the newer and easier to use wiringPiISR() function below.

- **int waitForInterrupt (int pin, int timeOut) ;**

When called, it will wait for an interrupt event to happen on that pin and your program will be stalled. The **timeOut** parameter is given in milliseconds, or can be -1 which means to wait forever.

The return value is -1 if an error occurred (and *errno* will be set appropriately), 0 if it timed out, or 1 on a successful interrupt event.

Before you call waitForInterrupt, you must first initialise the GPIO pin and at present the only way to do this is to use the **gpio** program, either in a script, or using the system() call from inside your program.

e.g. We want to wait for a falling-edge interrupt on GPIO pin 0, so to setup the hardware, we need to run:

```
gpio edge 0 falling
```

before running the program.

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

This function registers a function to received interrupts on the specified pin. The edgeType parameter is either **INT_EDGE_FALLING**, **INT_EDGE_RISING**, **INT_EDGE_BOTH** or **INT_EDGE_SETUP**. If it is **INT_EDGE_SETUP** then no initialisation of the pin will happen – it's assumed that you have already setup the pin elsewhere (e.g. with the ***gpio*** program), but if you specify one of the other types, then the pin will be exported and initialised as specified. This is accomplished via a suitable call to the ***gpio*** utility program, so it need to be available.

The pin number is supplied in the current mode – native wiringPi, BCM_GPIO, physical or Sys modes.

This function will work in any mode, and does not need root privileges to work.

The function will be called when the interrupt triggers. When it is triggered, it's cleared in the dispatcher before calling your function, so if a subsequent interrupt fires before you finish your handler, then it won't be missed. (However it can only track one more interrupt, if more than one interrupt fires while one is being handled then they will be ignored)

This function is run at a high priority (if the program is run using sudo, or as root) and executes concurrently with the main program. It has full access to all the global variables, open file handles and so on.

See the *isr.c* example program for more details on how to use this feature.

Concurrent Processing (multi-threading)

wiringPi has a simplified interface to the Linux implementation of Posix threads, as well as a (simplified) mechanisms to access mutex's (Mutual exclusions)

Using these functions you can create a new process (a function inside your main program) which runs concurrently with your main program and using the mutex mechanisms, safely pass variables between them.

- **int piThreadCreate (*name*) ;**

This function creates a thread which is another function in your program previously declared using the **PI_THREAD** declaration. This function is then run concurrently with your main program. An example may be

to have this function wait for an interrupt while your program carries on doing other tasks. The thread can indicate an event, or action by using global variables to communicate back to the main program, or other threads.

Thread functions are declared as follows:

```
PI_THREAD (myThread)
{
  .. code here to run concurrently with
        the main program, probably in an
        infinite loop
}
```

and would be started in the main program with:

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

This is really nothing more than a simplified interface to the Posix threads mechanism that Linux supports. See the manual pages on Posix threads (man pthread) if you need more control over them.

- **piLock (int keyNum) ;**
- **piUnlock (int keyNum) ;**

These allow you to synchronise variable updates from your main program to any threads running in your program. keyNum is a number from 0 to 3 and represents a “key”. When another process tries to lock the same key, it will be stalled until the first process has unlocked the same key.

You may need to use these functions to ensure that you get valid data when exchanging data between your main program and a thread – otherwise it’s possible that the thread could wake-up halfway during your data copy and change the data – so the data you end up copying is incomplete, or invalid. See the wfi.c program in the examples directory for an example.