gpio via C

Accessing **/sys** through C

Using **usleep()**

togglegpio.c

```
// Blink pin 3 on port B at 1 Hz
// Just add an LED and see the light! ;)
//
//Created by Dingo_aus, 7 January 2009
//email: dingo_aus [at] internode <dot> on /dot/ net
// From
http://www.avrfreaks.net/wiki/index.php/Documentation:Linux/GPIO#gpio_framework
//
//Created in AVR32 Studio (version 2.0.2) running on Ubuntu 8.04
// Modified by Mark A. Yoder, 21-July-2011
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#define SYSFS_GPIO_DIR "/sys/class/gpio"

FILE *fp;
```

togglegpio.c (2)

```
int main(int argc, char** argv)
{
    //create a variable to store whether we are sending a '1' or a '0'
    char set_value[5];
    //Integer to keep track of whether we want on or off
    int toggle = 0;
    int onOffTime;          // Time in micro sec to keep the signal on/off

    if (argc < 2) {
         printf("Usage: %s <on/off time in us>\n\n", argv[0]);
         printf("Waits for a change in the GPIO pin voltage level or input on stdin\n");
         exit(-1);
    }
    onOffTime = atoi(argv[1]);
    printf("\n**********************************\n"
         "*  Welcome to PIN Blink program  *\n"
         "*  ....blinking gpio 130         *\n"
         "*  ....period of %d us.........*\n"
         "**********************************\n", 2*onOffTime);
```

togglegpio.c (3)

```
//Using sysfs we need to write the 3 digit gpio number to
     /sys/class/gpio/export
//This will create the folder /sys/class/gpio/gpio130
if ((fp = fopen(SYSFS_GPIO_DIR "/export", "ab")) == NULL) {
            printf("Cannot open export file.\n");
            exit(1);
     }
//Set pointer to beginning of the file
     rewind(fp);
     //Write our value of "130" to the file
     strcpy(set_value, "130");
     fwrite(&set_value, sizeof(char), 3, fp);
     fclose(fp);

printf("...export file accessed, new pin now accessible\n");
```

togglegpio.c (4)

```
//SET DIRECTION
//Open the LED's sysfs file in binary for reading and
   writing, store file pointer in fp
if ((fp = fopen(SYSFS_GPIO_DIR "/gpio130/direction",
   "rb+")) == NULL) {
   printf("Cannot open direction file.\n");
   exit(1);
}
//Set pointer to begining of the file
rewind(fp);
//Write our value of "out" to the file
strcpy(set_value,"out");
fwrite(&set_value, sizeof(char), 3, fp);
fclose(fp);
printf("...direction set to output\n");
```

togglegpio.c (5)

```
if ((fp = fopen(SYSFS_GPIO_DIR "/gpio130/value",
   "rb+")) == NULL) {
   printf("Cannot open value file.\n");
   exit(1);
}
```

togglegpio.c (6)

```
   //Run an infinite loop - will require Ctrl-C to
exit        this program
   while(1) {
      toggle = !toggle;
      if(toggle) {
         //Set pointer to begining of the file
         rewind(fp);
         //Write our value of "1" to the file
         strcpy(set_value,"1");
         fwrite(&set_value, sizeof(char), 1, fp);
         fflush(fp);
//       printf("...value set to 1...\n");
      }
```

togglegpio.c (7)

```
      else {
         //Set pointer to begining of the file
         rewind(fp);
         //Write our value of "0" to the file
         strcpy(set_value,"0");
         fwrite(&set_value, sizeof(char), 1, fp);
         fflush(fp);
//       printf("...value set to 0...\n");
      }
      //Pause for a while
      usleep(onOffTime);
   }
   fclose(fp);
   return 0;
}
```

gpio via C

Accessing **/sys** through C

Using **poll()**

gpio-int-test.c

```
/****************************************************************
 * Main
 ****************************************************************/
int main(int argc, char **argv, char **envp)
{
   struct pollfd fdset[2];
   int nfds = 2;
   int gpio_fd, timeout, rc;
   char *buf[MAX_BUF];
   unsigned int gpio;
   int len;

   if (argc < 2) {
      printf("Usage: gpio-int <gpio-pin>\n\n");
      printf("Waits for a change in the GPIO pin voltage level or input on stdin\n");
      exit(-1);
   }
```

gpio-int-test.c (2)

```
   // Set the signal callback for Ctrl-C
   signal(SIGINT, signal_handler);

   gpio = atoi(argv[1]);

   gpio_export(gpio);
   gpio_set_dir(gpio, 0);
// Can be rising, falling or both
   gpio_set_edge(gpio, "both");
   gpio_fd = gpio_fd_open(gpio);

   timeout = POLL_TIMEOUT;
```

signal_handler()

```
/****************************************************************
 * Global variables
 ****************************************************************/
int keepgoing = 1;    // Set to 0 when ctrl-c is pressed

/****************************************************************
 * signal_handler
 ****************************************************************/
// Callback called when SIGINT is sent to the process (Ctrl-C)
void signal_handler(int sig)
{
   printf( "Ctrl-C pressed, cleaning up and exiting..\n" );
   keepgoing = 0;
}
```

gpio-int-test.c (3)

```
    while (keepgoing) {
        memset((void*)fdset, 0, sizeof(fdset));

        fdset[0].fd = STDIN_FILENO;
        fdset[0].events = POLLIN;
        fdset[1].fd = gpio_fd;
        fdset[1].events = POLLPRI;

        rc = poll(fdset, nfds, timeout);

        if (rc < 0) {
            printf("\npoll() failed!\n");
            return -1;
        }
        if (rc == 0) {
            printf(".");
        }
```

gpio-int-test.c (4)

```
        if (fdset[1].revents & POLLPRI) {
            // Read from the start of the file
            lseek(fdset[1].fd, 0, SEEK_SET);
            len = read(fdset[1].fd, buf, MAX_BUF);
            printf("\npoll() GPIO %d interrupt occurred,
                value=%c, len=%d\n", gpio, buf[0], len);
            }
        if (fdset[0].revents & POLLIN) {
            (void)read(fdset[0].fd, buf, 1);
            printf("\npoll() stdin read 0x%2.2X\n",
                               (unsigned int) buf[0]);
            }
            fflush(stdout);
        } // End of while(keepgoing)
        gpio_fd_close(gpio_fd);
        return 0;
    }
```