```c
/*
,* Copyright © 2013, Malcolm Sparks <malcolm@congreve.com>. All Rights Reserved.
,*
,* A program to convert USB firing events from the Dream Cheeky 'Big Red Button' to MQTT events.
,*/

#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define LID_CLOSED 21
#define BUTTON_PRESSED 22
#define LID_OPEN 23

int main(int argc, char **argv)
{
  int fd;
  int i, res, desc_size = 0;
  char buf[256];

  /* Use a udev rule to make this device */
  fd = open("/dev/big_red_button", O_RDWR|O_NONBLOCK);

  if (fd < 0) {
    perror("Unable to open device");
    return 1;
  }

  int prior = LID_CLOSED;

  while (1) {
    memset(buf, 0x0, sizeof(buf));
    buf[0] = 0x08;
    buf[7] = 0x02;

    res = write(fd, buf, 8);
    if (res < 0) {
      perror("write");
      exit(1);
    }

    memset(buf, 0x0, sizeof(buf));
    res = read(fd, buf, 8);

    if (res >= 0) {
      if (prior == LID_CLOSED && buf[0] == LID_OPEN) {
         printf("Ready to fire!\n");
         fflush(stdout);
      } else if (prior != BUTTON_PRESSED && buf[0] == BUTTON_PRESSED) {
        printf("Fire!\n");
        fflush(stdout);
      } else if (prior != LID_CLOSED && buf[0] == LID_CLOSED) {
        printf("Stand down!\n");
        fflush(stdout);
      }
      prior = buf[0];
    }
    usleep(20000); /* Sleep for 20ms*/
  }
}
```

```
all:    pcsensor

CFLAGS = -O2 -Wall

pcsensor:   pcsensor.c
    ${CC} -DUNIT_TEST -o $@ $^ -lusb

clean:
    rm -f pcsensor *.o

rules-install:          # must be superuser to do this
    cp ../udev/99-tempsensor.rules /etc/udev/rules.d
```

```c
/*
 * pcsensor.c by Philipp Adelt (c) 2012 (info@philipp.adelt.net)
 * based on Juan Carlos Perez (c) 2011 (cray@isp-sl.com)
 * based on Temper.c by Robert Kavaler (c) 2009 (relavak.com)
 * All rights reserved.
 *
 * Temper driver for linux. This program can be compiled either as a library
 * or as a standalone program (-DUNIT_TEST). The driver will work with some
 * TEMPer usb devices from RDing (www.PCsensor.com).
 *
 * This driver works with USB devices presenting ID 0c45:7401.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *     * Redistributions of source code must retain the above copyright
 *       notice, this list of conditions and the following disclaimer.
 *
 * THIS SOFTWARE IS PROVIDED BY Philipp Adelt (and other contributors) ''AS IS'' AND ANY
 * EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL Philipp Adelt (or other contributors) BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */



#include <usb.h>
#include <stdio.h>
#include <time.h>

#include <string.h>
#include <errno.h>
#include <signal.h>


#define VERSION "1.0.0"

#define VENDOR_ID  0x0c45
#define PRODUCT_ID 0x7401

#define INTERFACE1 0x00
#define INTERFACE2 0x01

const static int reqIntLen=8;
const static int reqBulkLen=8;
const static int endpoint_Int_in=0x82; /* endpoint 0x81 address for IN */
const static int endpoint_Int_out=0x00; /* endpoint 1 address for OUT */
const static int endpoint_Bulk_in=0x82; /* endpoint 0x81 address for IN */
const static int endpoint_Bulk_out=0x00; /* endpoint 1 address for OUT */
const static int timeout=5000; /* timeout in ms */

const static char uTemperatura[] = { 0x01, 0x80, 0x33, 0x01, 0x00, 0x00, 0x00, 0x00 };
const static char uIni1[] = { 0x01, 0x82, 0x77, 0x01, 0x00, 0x00, 0x00, 0x00 };
const static char uIni2[] = { 0x01, 0x86, 0xff, 0x01, 0x00, 0x00, 0x00, 0x00 };

static int bsalir=1;
static int debug=0;
static int seconds=5;
static int formato=0;
static int mrtg=0;
static int calibration=0;


void bad(const char *why) {
        fprintf(stderr,"Fatal error> %s\n",why);
        exit(17);
}



usb_dev_handle *find_lvr_winusb();

void usb_detach(usb_dev_handle *lvr_winusb, int iInterface) {
        int ret;

    ret = usb_detach_kernel_driver_np(lvr_winusb, iInterface);
    if(ret) {
        if(errno == ENODATA) {
            if(debug) {
                printf("Device already detached\n");
            }
        } else {
            if(debug) {
                printf("Detach failed: %s[%d]\n",
                       strerror(errno), errno);
                printf("Continuing anyway\n");
            }
        }
    } else {
        if(debug) {
            printf("detach successful\n");
        }
    }
}

usb_dev_handle* setup_libusb_access(int devicenum) {
     usb_dev_handle *lvr_winusb;

     if(debug) {
        usb_set_debug(255);
     } else {
        usb_set_debug(0);
    }
    usb_init();
    usb_find_busses();
    usb_find_devices();


    if(!(lvr_winusb = find_lvr_winusb(devicenum))) {
                printf("Couldn't find the USB device, Exiting\n");
                return NULL;
        }


        usb_detach(lvr_winusb, INTERFACE1);


        usb_detach(lvr_winusb, INTERFACE2);


        if (usb_set_configuration(lvr_winusb, 0x01) < 0) {
                printf("Could not set configuration 1\n");
                return NULL;
        }


        // Microdia tiene 2 interfaces
        if (usb_claim_interface(lvr_winusb, INTERFACE1) < 0) {
                printf("Could not claim interface\n");
                return NULL;
        }

        if (usb_claim_interface(lvr_winusb, INTERFACE2) < 0) {
                printf("Could not claim interface\n");
                return NULL;
        }

        return lvr_winusb;
}



usb_dev_handle *find_lvr_winusb(int devicenum) {
        // iterates to the devicenum'th device for installations with multiple sensors
        struct usb_bus *bus;
        struct usb_device *dev;

        for (bus = usb_busses; bus; bus = bus->next) {
        for (dev = bus->devices; dev; dev = dev->next) {
                        if (dev->descriptor.idVendor == VENDOR_ID &&
                                dev->descriptor.idProduct == PRODUCT_ID ) {
                                if (devicenum>0) {
                                  devicenum--;
                                  continue;
```

```
                                        }
                                        usb_dev_handle *handle;
                                        if(debug) {
                                          printf("lvr_winusb with Vendor Id: %x and Product Id: %x
                                          found.\n", VENDOR_ID, PRODUCT_ID);
                                        }

                                        if (!(handle = usb_open(dev))) {
                                                printf("Could not open USB device\n");
                                                return NULL;
                                        }
                                        return handle;
                                }
                        }
                }
                return NULL;
}


void ini_control_transfer(usb_dev_handle *dev) {
    int r,i;

    char question[] = { 0x01,0x01 };

    r = usb_control_msg(dev, 0x21, 0x09, 0x0201, 0x00, (char *) question, 2, timeout);
    if( r < 0 )
    {
          perror("USB control write"); bad("USB write failed");
    }


    if(debug) {
      for (i=0;i<reqIntLen; i++) printf("%02x ",question[i] & 0xFF);
      printf("\n");
    }
}

void control_transfer(usb_dev_handle *dev, const char *pquestion) {
    int r,i;

    char question[reqIntLen];

    memcpy(question, pquestion, sizeof question);

    r = usb_control_msg(dev, 0x21, 0x09, 0x0200, 0x01, (char *) question, reqIntLen, timeout);
    if( r < 0 )
    {
          perror("USB control write"); bad("USB write failed");
    }

    if(debug) {
        for (i=0;i<reqIntLen; i++) printf("%02x ",question[i]  & 0xFF);
        printf("\n");
    }
}

void interrupt_transfer(usb_dev_handle *dev) {

    int r,i;
    char answer[reqIntLen];
    char question[reqIntLen];
    for (i=0;i<reqIntLen; i++) question[i]=i;
    r = usb_interrupt_write(dev, endpoint_Int_out, question, reqIntLen, timeout);
    if( r < 0 )
    {
          perror("USB interrupt write"); bad("USB write failed");
    }
    r = usb_interrupt_read(dev, endpoint_Int_in, answer, reqIntLen, timeout);
    if( r != reqIntLen )
    {
          perror("USB interrupt read"); bad("USB read failed");
    }

    if(debug) {
       for (i=0;i<reqIntLen; i++) printf("%i, %i, \n",question[i],answer[i]);
    }

    usb_release_interface(dev, 0);
}

void interrupt_read(usb_dev_handle *dev) {

    int r,i;
    unsigned char answer[reqIntLen];
    bzero(answer, reqIntLen);

    r = usb_interrupt_read(dev, 0x82, answer, reqIntLen, timeout);
    if( r != reqIntLen )
    {
          perror("USB interrupt read"); bad("USB read failed");
    }

    if(debug) {
       for (i=0;i<reqIntLen; i++) printf("%02x ",answer[i]  & 0xFF);

       printf("\n");
    }
}

void interrupt_read_temperatura(usb_dev_handle *dev, float *tempC) {

    int r,i, temperature;
    unsigned char answer[reqIntLen];
    bzero(answer, reqIntLen);

    r = usb_interrupt_read(dev, 0x82, answer, reqIntLen, timeout);
    if( r != reqIntLen )
    {
          perror("USB interrupt read"); bad("USB read failed");
    }


    if(debug) {
      for (i=0;i<reqIntLen; i++) printf("%02x ",answer[i]  & 0xFF);

      printf("\n");
    }

    temperature = (answer[3] & 0xFF) + (answer[2] << 8);
    temperature += calibration;
    *tempC = temperature * (125.0 / 32000.0);

}

void bulk_transfer(usb_dev_handle *dev) {

    int r,i;
    char answer[reqBulkLen];

    r = usb_bulk_write(dev, endpoint_Bulk_out, NULL, 0, timeout);
    if( r < 0 )
    {
          perror("USB bulk write"); bad("USB write failed");
    }
    r = usb_bulk_read(dev, endpoint_Bulk_in, answer, reqBulkLen, timeout);
    if( r != reqBulkLen )
    {
          perror("USB bulk read"); bad("USB read failed");
    }


    if(debug) {
      for (i=0;i<reqBulkLen; i++) printf("%02x ",answer[i]  & 0xFF);
    }

    usb_release_interface(dev, 0);
}


void ex_program(int sig) {
      bsalir=1;

      (void) signal(SIGINT, SIG_DFL);
}

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

     usb_dev_handle *lvr_winusb = NULL;
     float tempc;
     int c;
     struct tm *local;
     time_t t;
     int devicenum = 0;

     while ((c = getopt (argc, argv, "mfcvhn:l::a:")) != -1)
     switch (c)
       {
       case 'v':
         debug = 1;
         break;
       case 'n':
         if (optarg != NULL) {
           if (!sscanf(optarg,"%i",&devicenum)==1) {
             fprintf (stderr, "Error: '%s' is not numeric.\n", optarg);
             exit(EXIT_FAILURE);
           }
         }
         break;
       case 'c':
         formato=1; //Celsius
         break;
       case 'f':
         formato=2; //Fahrenheit
         break;
       case 'm':
         mrtg=1;
         break;
       case 'l':
         if (optarg!=NULL){
           if (!sscanf(optarg,"%i",&seconds)==1) {
             fprintf (stderr, "Error: '%s' is not numeric.\n", optarg);
             exit(EXIT_FAILURE);
           } else {
              bsalir = 0;
              break;
           }
         } else {
           bsalir = 0;
           seconds = 5;
           break;
         }
       case 'a':
         if (!sscanf(optarg,"%i",&calibration)==1) {
             fprintf (stderr, "Error: '%s' is not numeric.\n", optarg);
             exit(EXIT_FAILURE);
         } else {
              break;
         }
       case '?':
       case 'h':
         printf("pcsensor version %s\n",VERSION);
     printf("      Aviable options:\n");
     printf("          -h help\n");
     printf("          -v verbose\n");
     printf("          -n[i] use device number i (0 is the first one found on the bus)\n");
     printf("          -l[n] loop every 'n' seconds, default value is 5s\n");
     printf("          -c output only in Celsius\n");
     printf("          -f output only in Fahrenheit\n");
     printf("          -a[n] increase or decrease temperature in 'n' degrees for device
     calibration\n");
     printf("          -m output for mrtg integration\n");

     exit(EXIT_FAILURE);
       default:
         if (isprint (optopt))
           fprintf (stderr, "Unknown option `-%c'.\n", optopt);
         else
           fprintf (stderr,
                    "Unknown option character `\\x%x'.\n",
                    optopt);
         exit(EXIT_FAILURE);
       }

     if (optind < argc) {
        fprintf(stderr, "Non-option ARGV-elements, try -h for help.\n");
        exit(EXIT_FAILURE);
     }

     if ((lvr_winusb = setup_libusb_access(devicenum)) == NULL) {
         exit(EXIT_FAILURE);
     }

     (void) signal(SIGINT, ex_program);

     ini_control_transfer(lvr_winusb);

     control_transfer(lvr_winusb, uTemperatura );
     interrupt_read(lvr_winusb);

     control_transfer(lvr_winusb, uIni1 );
     interrupt_read(lvr_winusb);

     control_transfer(lvr_winusb, uIni2 );
     interrupt_read(lvr_winusb);
     interrupt_read(lvr_winusb);



     do {
           control_transfer(lvr_winusb, uTemperatura );
           interrupt_read_temperatura(lvr_winusb, &tempc);

           t = time(NULL);
           local = localtime(&t);
```

```
            if (mrtg) {
               if (formato==2) {
                   printf("%.2f\n", (9.0 / 5.0 * tempc + 32.0));
                   printf("%.2f\n", (9.0 / 5.0 * tempc + 32.0));
               } else {
                   printf("%.2f\n", tempc);
                   printf("%.2f\n", tempc);
               }

               printf("%02d:%02d\n",
                           local->tm_hour,
                           local->tm_min);

               printf("pcsensor\n");
            } else {
               printf("%04d/%02d/%02d %02d:%02d:%02d ",
                           local->tm_year +1900,
                           local->tm_mon + 1,
                           local->tm_mday,
                           local->tm_hour,
                           local->tm_min,
                           local->tm_sec);

               if (formato==2) {
                   printf("Temperature %.2fF\n", (9.0 / 5.0 * tempc + 32.0));
               } else if (formato==1) {
                   printf("Temperature %.2fC\n", tempc);
               } else {
                   printf("Temperature %.2fF %.2fC\n", (9.0 / 5.0 * tempc + 32.0), tempc);
               }
            }

            if (!bsalir)
               sleep(seconds);
      } while (!bsalir);

      usb_release_interface(lvr_winusb, INTERFACE1);
      usb_release_interface(lvr_winusb, INTERFACE2);

      usb_close(lvr_winusb);

      return 0;
}
```