```
/** @file linux/rs485.c  Provides Linux-specific functions for RS-485 serial.
*/

/* The module handles sending data out the RS-485 port */
/* and handles receiving data from the RS-485 port. */
/* Customize this file for your specific hardware */
#include <errno.h>
#include <stddef.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Linux includes */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <unistd.h>
#include <sched.h>

/* Local includes */
#include "mstp.h"
#include "rs485.h"
#include "fifo.h"

#include <pthread.h>
#include <sys/select.h>
#include <sys/time.h>


/* handle returned from open() */
static int RS485_Handle = -1;
/* baudrate settings are defined in <asm/termbits.h>, which is
   included by <termios.h> */
static unsigned int RS485_Baud = B38400;
/* serial port name, /dev/ttyS0,
  /dev/ttyUSB0 for USB->RS485 from B&B Electronics USOPTL4 */
static char *RS485_Port_Name = "/dev/ttyUSB0";
/* some terminal I/O have RS-485 specific functionality */
#ifndef RS485MOD
#define RS485MOD 0
#endif
/* serial I/O settings */
static struct termios RS485_oldtio;

/* Ring buffer for incoming bytes, in order to speed up the receiving. */
static FIFO_BUFFER Rx_FIFO;
/* buffer size needs to be a power of 2 */
static uint8_t Rx_Buffer[1 << 12];

static pthread_mutex_t Reader_Mutex, IOMutex;

#define _POSIX_SOURCE 1 /* POSIX compliant source */
```

```
static void *rs485_read_task(
    void *arg)
{
    uint8_t buf[1 << 11];
    int count, n;
    fd_set input;
    struct timeval cekalica;
    FD_ZERO(&input);
    FD_SET(RS485_Handle, &input);
    cekalica.tv_sec = 1;
    cekalica.tv_usec = 0;
    for (;;) {
        n = select(RS485_Handle + 1, &input, NULL, NULL, &cekalica);
        if (n < 0) {
            continue;
        }
        if (FD_ISSET(RS485_Handle, &input)) {
            pthread_mutex_lock(&IOMutex);
            count = read(RS485_Handle, buf, sizeof(buf));
            pthread_mutex_unlock(&IOMutex);
            if (count > 0) {
                pthread_mutex_lock(&Reader_Mutex);
                FIFO_Add(&Rx_FIFO, &buf[0], count);
                pthread_mutex_unlock(&Reader_Mutex);
            }
            usleep(5000);
        }
        FD_SET(RS485_Handle, &input);
        cekalica.tv_sec = 1;
        cekalica.tv_usec = 0;
    }
    return NULL;
}

/*************************************************************************
* DESCRIPTION: Configures the interface name
* RETURN:      none
* ALGORITHM:   none
* NOTES:       none
*************************************************************************/
void RS485_Set_Interface(
    char *ifname)
{
    /* note: expects a constant char, or char from the heap */
    if (ifname) {
        RS485_Port_Name = ifname;
    }
}

/*************************************************************************
* DESCRIPTION: Returns the interface name
*************************************************************************/
const char *RS485_Interface(
    void)
{
    return RS485_Port_Name;
}
```

```
/*************************************************************************
* DESCRIPTION: Returns the baud rate that we are currently running at
* RETURN:      none
*************************************************************************
/
uint32_t RS485_Get_Baud_Rate(
    void)
{
    switch (RS485_Baud) {
        case B19200:
            return 19200;
        case B38400:
            return 38400;
        case B57600:
            return 57600;
        case B115200:
            return 115200;
        default:
        case B9600:
            return 9600;
    }
}

/*************************************************************************
* DESCRIPTION: Sets the baud rate for the chip USART
* RETURN:      none
*************************************************************************
/
bool RS485_Set_Baud_Rate(
    uint32_t baud)
{
    bool valid = true;

    switch (baud) {
        case 9600:
            RS485_Baud = B9600;
            break;
        case 19200:
            RS485_Baud = B19200;
            break;
        case 38400:
            RS485_Baud = B38400;
            break;
        case 57600:
            RS485_Baud = B57600;
            break;
        case 115200:
            RS485_Baud = B115200;
            break;
        default:
            valid = false;
            break;
    }

    if (valid) {
        /* FIXME: store the baud rate */
    }
```

```
    return valid;
}

/* Transmits a Frame on the wire */
void RS485_Send_Frame(
    volatile struct mstp_port_struct_t *mstp_port,      /* port specific data
*/
    uint8_t * buffer,   /* frame to send (up to 501 bytes of data) */
    uint16_t nbytes)
{       /* number of bytes of data (up to 501) */
    ssize_t written = 0;
    int greska;
    /*
       On  success,  the  number of bytes written are returned (zero
indicates
       nothing was written).  On error, -1  is  returned,  and  errno  is
set
       appropriately.   If  count  is zero and the file descriptor refers to
a
       regular file, 0 will be returned without causing any other effect.
For
       a special file, the results are not portable.
     */
    pthread_mutex_lock(&IOMutex);
    written = write(RS485_Handle, buffer, nbytes);
    pthread_mutex_unlock(&IOMutex);
    greska = errno;
    if (written <= 0)
        printf("write error: %s\n", strerror(greska));
    /*  tcdrain(RS485_Handle); */
    /* per MSTP spec, sort of */
    if (mstp_port) {
        mstp_port->SilenceTimerReset();
    }

    return;
}

/* called by timer, interrupt(?) or other thread */
void RS485_Check_UART_Data(
    volatile struct mstp_port_struct_t *mstp_port)
{
    if (mstp_port->ReceiveError == true) {
        /* wait for state machine to clear this */
        /*mstp_port->ReceiveError=false; */
        return;
    }
    /* wait for state machine to read from the DataRegister */
    /*else */
    if (mstp_port->DataAvailable == false) {
        /* check for data */
        pthread_mutex_lock(&Reader_Mutex);
        if (FIFO_Count(&Rx_FIFO) > 0) {
            mstp_port->DataRegister = FIFO_Get(&Rx_FIFO);
            mstp_port->DataAvailable = true;
        }
```

```
        pthread_mutex_unlock(&Reader_Mutex);
    }
}

void RS485_Cleanup(
    void)
{
    /* restore the old port settings */
    tcsetattr(RS485_Handle, TCSANOW, &RS485_oldtio);
    close(RS485_Handle);
    pthread_mutex_destroy(&Reader_Mutex);
    pthread_mutex_destroy(&IOMutex);
}


void RS485_Initialize(
    void)
{
    struct termios newtio;
    unsigned long hThread = 0;
    printf("RS485: Initializing %s", RS485_Port_Name);
    /*
       Open device for reading and writing.
       Blocking mode - more CPU effecient
     */
    RS485_Handle = open(RS485_Port_Name, O_RDWR | O_NOCTTY /*| O_NDELAY */ );
    if (RS485_Handle < 0) {
        perror(RS485_Port_Name);
        exit(-1);
    }
#if 0
    /* non blocking for the read */
    fcntl(RS485_Handle, F_SETFL, FNDELAY);
#else
    /* efficient blocking for the read */
    fcntl(RS485_Handle, F_SETFL, 0);
#endif
    /* save current serial port settings */
    tcgetattr(RS485_Handle, &RS485_oldtio);
    /* clear struct for new port settings */
    bzero(&newtio, sizeof(newtio));
    /*
       BAUDRATE: Set bps rate. You could also use cfsetispeed and
cfsetospeed.
       CRTSCTS : output hardware flow control (only used if the cable has
       all necessary lines. See sect. 7 of Serial-HOWTO)
       CS8     : 8n1 (8bit,no parity,1 stopbit)
       CLOCAL  : local connection, no modem contol
       CREAD   : enable receiving characters
     */
    newtio.c_cflag = RS485_Baud | CS8 | CLOCAL | CREAD | RS485MOD;
    /* Raw input */
    newtio.c_iflag = 0;
    /* Raw output */
    newtio.c_oflag = 0;
    /* no processing */
    newtio.c_lflag = 0;
```

```
    /* activate the settings for the port after flushing I/O */
    tcsetattr(RS485_Handle, TCSAFLUSH, &newtio);
    /* destructor */
    atexit(RS485_Cleanup);
    /* flush any data waiting */
    usleep(200000);
    tcflush(RS485_Handle, TCIOFLUSH);
    /* ringbuffer */
    FIFO_Init(&Rx_FIFO, Rx_Buffer, sizeof(Rx_Buffer));
    pthread_mutex_init(&Reader_Mutex, NULL);
    pthread_mutex_init(&IOMutex, NULL);
    pthread_create(&hThread, NULL, rs485_read_task, NULL);
    printf("=success!\n");
}

#ifdef TEST_RS485
#include <string.h>
int main(
    int argc,
    char *argv[])
{
    uint8_t buf[8];
    char *wbuf = { "BACnet!" };
    size_t wlen = strlen(wbuf) + 1;
    unsigned i = 0;
    size_t written = 0;
    int rlen;

    /* argv has the "/dev/ttyS0" or some other device */
    if (argc > 1) {
        RS485_Set_Interface(argv[1]);
    }
    RS485_Set_Baud_Rate(38400);
    RS485_Initialize();

    for (;;) {
        written = write(RS485_Handle, wbuf, wlen);
        rlen = read(RS485_Handle, buf, sizeof(buf));
        /* print any characters received */
        if (rlen > 0) {
            for (i = 0; i < rlen; i++) {
                fprintf(stderr, "%02X ", buf[i]);
            }
        }
    }

    return 0;
}
#endif
```