/* Port485.c */

```
#include "Port485.h"
#include <termios.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h> /* For Memset*/

/* Configures the ttyS1 port for RS485*/
/* Returns the file descriptor for the port */

int SetTerm(void)
{

  int fd;
  struct termios newtio;
  /*
    Open modem device for reading and writing and not as controlling tty
    because we don't want to get killed if linenoise sends CTRL-C.
  */
  fd = open(SERIALDEVICE, O_RDWR | O_NOCTTY );
  if (fd <0) {perror(SERIALDEVICE); exit(EXIT_FAILURE); }

  memset(&newtio,0,sizeof(newtio)); /* clear struct for new port settings */

  /*
    BAUDRATE: Set bps rate. You could also use cfsetispeed and cfsetospeed.
    CS8     : 8n1 (8bit,no parity,1 stopbit)
    CLOCAL  : local connection, no modem contol
    CREAD   : enable receiving characters
  */
  newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;

  /*
   IGNPAR  : ignore bytes with parity errors
   ICRNL   : map CR to NL (otherwise a CR input on the other computer
           will not terminate input)
                   otherwise make device raw (no other input processing)
  */
  newtio.c_iflag = IGNPAR | ICRNL;

  /*
   Raw output.
  */
  newtio.c_oflag = 0;

  /*
   ICANON  : enable canonical input
   disable all echo functionality, and don't send signals to calling program
  */
  newtio.c_lflag = ICANON;

  /*
    initialize some control characters
    default values can be found in /usr/include/termios.h, and are given
    in the comments, but we don't need them here
  */
  newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
  newtio.c_cc[VTIME]    = 0;     /* inter-character timer unused */
  newtio.c_cc[VMIN]     = 1;     /* blocking read until 1 character arrives */

  /*
    now clear the line and activate the settings for the port
  */
  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio);

  /* All done */

  return fd;
}
```

```
/* Port485.h */
#ifndef _PORT485H_
#define _PORT485H_

/* baudrate settings are defined in <asm/termbits.h>, which is
   included by <termios.h> */

#include <termios.h>

#define BAUDRATE B9600
#define SERIALDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX compliant source */

int SetTerm(void);

#endif

/*
 * stk.c: Socket Toolkit
 * Copyright (c) 2003 Philip Bock
 */

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

#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>

#include "stk.h"
#include "fixm68k.h"

struct sockaddr *stk_address(const char *hostname, int port)
{
	struct hostent *host;
	struct sockaddr_in *address;

	if ((host = gethostbyname(hostname)) == NULL)
		return NULL;

	address = malloc(sizeof(struct sockaddr_in));

	address->sin_family = AF_INET;
	address->sin_port = htons(port);
	memcpy(&address->sin_addr, host->h_addr_list[0], host->h_length);
	memset(&address->sin_zero, '\0', 8);

	return (struct sockaddr *) address;
}

int stk_connect(const char *hostname, int port, int type)
{
	struct sockaddr *address;
	int sckt;

	if ((address = stk_address(hostname, port)) == NULL)
		return -2;

	if ((sckt = socket(PF_INET, type, 0)) == -1)
		return -1;

	if (connect(sckt, address, sizeof(struct sockaddr)) == -1)
	{
		close(sckt);
		return -3;
	}

	free(address);

	return sckt;
}
```

```
int stk_listen(const char *ip, int port, int backlog)
{
	struct sockaddr *address;
	int sckt;

	if ((address = stk_address(ip, port)) == NULL)
		return -2;

	if ((sckt = socket(PF_INET, SOCK_STREAM, 0)) == -1)
		return -1;

	if (bind(sckt, address, sizeof(struct sockaddr)) == -1)
	{
		close(sckt);
		return -3;
	}

	free(address);

	if (listen(sckt, backlog) == -1)
	{
		close(sckt);
		return -4;
	}

	return sckt;
}

/*
 * stk.h: Socket Toolkit
 * Copyright (c) 2003 Philip Bock
 */

#ifndef __STK_H__
#define __STK_H__

struct sockaddr *stk_address(const char *hostname, int port);
int stk_connect(const char *hostname, int port, int type);
int stk_listen(const char *ip, int port, int backlog);

#endif
```