3. Файлы исходного кода программы

3.1 Файл main.c

```
#include "uip/uip.h"
#include "uip/uip_arp.h"
#include "uip/httpd.h"
#include "uip/timer.h"
#include "lpc17xx_emac.h"
#include "uart.h"

int main(void)
{
  int i;
  uip_ipaddr_t ipaddr;
  struct timer periodic_timer;

  timer_set(&periodic_timer, CLOCK_SECOND / 2);

  EMAC_Init(); // network_device_init
  uip_init();

  uip_ipaddr(ipaddr, 192,168,0,55);
  uip_sethostaddr(ipaddr);

  UART0Init();
  uip_listen(HTONS(23));

  while(1) {
    uip_len = network_device_read();
    if(uip_len > 0) {
      uip_input();
      if(uip_len > 0) {
        network_device_send();
      }
    } else if(timer_expired(&periodic_timer)) {
      timer_reset(&periodic_timer);
      for(i = 0; i < UIP_CONNS; i++) {
        uip_periodic(i);
        if(uip_len > 0) {
          network_device_send();
        }
      }
    }
  }
  return 0;
}
```

3.2 Файл uart.c

```
#include "LPC17xx.h"
#include "type.h"
#include "uart.h"

volatile uint32_t UART0Status;
uint8_t UART0SendBuffer[TXBUFSIZE], UART0RecvBuffer[RXBUFSIZE];
const uint8_t *txBufEnd = UART0SendBuffer + TXBUFSIZE, *rxBufEnd =
UART0RecvBuffer + RXBUFSIZE;
volatile uint8_t *UART0RBTail = UART0RecvBuffer, *UART0SBHead = UART0SendBuffer,
                 *UART0SBTail = UART0SendBuffer, UART0SBEmpty = 1, UART0RBEmpty
= 1;

void UART0PushSend( uint8_t *data, uint16_t length ) {
      uint16_t pos = 0;
      while( UART0SBHead != UART0SBTail || UART0SBEmpty ) {
            // UART0SendBuffer not full
            UART0SBEmpty = 0;
            *UART0SBTail++ = data[pos++];
            if( pos == length ) {
                  break;
            }
      }
      LPC_UART0->IER |= IER_THRE;
}

uint32_t UART0Init() {
      uint32_t Fdiv;
      uint32_t pclkdiv, pclk;

      LPC_PINCON->PINSEL0 &= ~0x000000F0;
      LPC_PINCON->PINSEL0 |= 0x00000050;  // RxD0 is P0.3 and TxD0 is P0.2
      /* By default, the PCLKSELx value is zero, thus, the PCLK for
      *           all the peripherals is 1/4 of the SystemFrequency. */
      pclkdiv = (LPC_SC->PCLKSEL0 >> 6) & 0x03; // Bit 6~7 is for UART0
      switch ( pclkdiv ) {
            case 0x00:
            default:
                  pclk = SystemFrequency/4;
                  break;
            case 0x01:
                  pclk = SystemFrequency;
                  break;
            case 0x02:
                  pclk = SystemFrequency/2;
                  break;
            case 0x03:
                  pclk = SystemFrequency/8;
                  break;
      }

      LPC_UART0->LCR = 0x03;              /* 8 bits, no parity, 1 stop bit */
      Fdiv = ( pclk / 16 ) / UART0BAUDRATE;          /* baud rate */
      LPC_UART0->DLM = Fdiv / 256;
      LPC_UART0->DLL = Fdiv % 256;
      LPC_UART0->LCR = 0x03;              /* DLAB = 0 */
      LPC_UART0->FCR = 0x07;              /* Enable and reset TX and RX FIFO. */
```

```
	NVIC_EnableIRQ(UART0_IRQn);
	LPC_UART0->IER = IER_RBR | IER_THRE | IER_RLS;	/* Enable UART0
interrupt */

	return (TRUE);
}

void UART0_IRQHandler() {
	uint8_t IIRValue;

	IIRValue = LPC_UART0->IIR;

	IIRValue >= 1;		// skip pending bit in IIR
	IIRValue &= 0x07;		// check bit 1~3, interrupt identification
	if ( IIRValue == IIR_RLS ) {
		// Receive Line Status
		uint8_t LSRValue;
		LSRValue = LPC_UART0->LSR;
		// Receive Line Status
		if ( LSRValue & ( LSR_OE | LSR_PE | LSR_FE | LSR_RXFE | LSR_BI ) ) {
			// There are errors or break interrupt
			// Read LSR will clear the interrupt
			UART0Status = LSRValue;
			uint8_t dummy = LPC_UART0->RBR;		// Dummy read on RX
to clear interrupt, then bail out
			return;
		}
		if ( LSRValue & LSR_RDR ) {
			// Receive Data Ready
			// If no error on RLS, normal ready, save into the data buffer.
			// Note: read RBR will clear the interrupt
			if ( UART0RBTail != rxBufEnd ) { // buffer not full
				*UART0RBTail = LPC_UART0->RBR;
				UART0RBTail++;
			}
			else {
				// send
			}
		}
	}
	else if ( IIRValue == IIR_RDA ) {
		// Receive Data Available
		if ( UART0RBTail != rxBufEnd ) { // buffer not full
			*UART0RBTail = LPC_UART0->RBR;
			UART0RBTail++;
		}
		else {
			// send
		}
	}
	else if ( IIRValue == IIR_CTI ) {
		// Character timeout indicator
		UART0Status |= 0x100;		// Bit 9 as the CTI error
	}
	else if ( IIRValue == IIR_THRE ) {
		// THRE interrupt
		uint8_t LSRValue = LPC_UART0->LSR;
		if ( LSRValue & LSR_THRE ) {
			if ( UART0SBTail != UART0SBHead ) { // Transmit FIFO not empty
```

```
                        LPC_UART1->THR = *UART0SBTail++;
                        if( UART0SBTail == txBufEnd ) {
                              UART0SBTail = UART0SendBuffer;
                        }
                  }
                  else {
                        UART0SBEmpty = 1;
                  }
            }
      }
}
```

3.2 Файл telnet.c

```
#include "uip/uip.h"
#include "telnet.h"
#include "uart.h"
#define TELNET_SB       4
#define TELNET_MODE_LINEMODE              34
#define TELNET_MODE_LINEMODE_EDIT         1
#define TELNET_MODE_LINEMODE_TRAPSIG      2
#define TELNET_SE       240
#define TELNET_WILL           251
#define TELNET_WONT           252
#define TELNET_DO       253
#define TELNET_DONT           254
#define TELNET_IAC            255
/*    ,--------------------.
 *   /    telnetState      |
 *  /   ,------------------'
 *  |  /  Action
 *  | -|--
 *  |0.|  < Send >
 *  |  |  IAC WILL LINEMODE
 *  |  |  255 251  34
 *  | -|--
 *  |0.|  < Wait for >
 *  |  |  IAC DO  LINEMODE
 *  |  |  255 253 34
 *  |  |  [0] [1] [2]
 *  | -|--
 *  |3.|  < Send >
 *  |  |  IAC SB  LINEMODE EDIT 0 IAC SE
 *  |  |  255 250 34       1    0 255 240
 *  | -|--
 *  |3.|  < Wait for >
 *  |  |  IAC SB  LINEMODE EDIT 0|ACK IAC SE
 *  |  |  255 250 34       1    4     255 240
 *  |  |  [3] [4] [5]      [6]  [7]   [8] [9]
 *  | -|--
 *  |10.  < Send >
 *  |  |  IAC SB  LINEMODE TRAPSIG 0 IAC SE
 *  |  |  255 250 34       2       0 255 240
 *  | -|--
 *  |10.  < Wait for >
 *  |  |  IAC SB  LINEMODE TRAPSIG 0|ACK IAC SE
 *  |  |  255 250 34       2       4     255 240
 *  |  |  [10][11][12]     [13]    [14]  [15][16]
 *  | -|--
 *  |17.  < Telnet setup complete >
 *  | -|--
 *  |128. < Connection established, idle >
 *  | -|--
 *  |129. < Connection established, tx data sent, waiting ack >
 *  |  |
 *   \/
 *
 */

const u8_t rxExpectedCount = 17, rxExpected[] = {
      255, 253, 34,
```

```
      255, 250, 34, 1, 4, 255, 240,
      255, 250, 34, 2, 4, 255, 240
};
u8_t telnetState = 0, rxExpectedByte = 0;

struct txDataType {
      u8_t data[TCPTXBUFSIZE];
      u8_t length;
} txData;

void connClosed() {
      telnetState = 0;
}

void telnetd_appcall(void) {
      if(uip_connected()) {
            // new connection
            // telnet setup
            telnetState = 0;
            txData.data[0] = TELNET_IAC;
            txData.data[1] = TELNET_WILL;
            txData.data[2] = TELNET_MODE_LINEMODE;
            txData.length = 3;
            uip_send( txData.data, txData.length );
      }

      if(uip_closed() ||
            uip_aborted() ||
            uip_timedout()) {
            connClosed();
      }

      if(uip_acked()) {
            if( telnetState == 129 ) {
                  telnetState = 128; // tcp ack
            }
      }

      if(uip_newdata()) {
            u8_t rxDataByte, rxDataLength = uip_datalen();
            if( telnetState & 128 ) {
                  // connection is good, pass recieved data to UART
                  UART0PushSend( uip_appdata, uip_datalen() );
            }
            else {
                  for( rxDataByte = 0; rxDataByte < rxDataLength && telnetState <
rxExpectedCount; ) {
                        if( ((uint8_t*)uip_appdata)[rxDataByte++] !=
rxExpected[telnetState] ) {
                              // unexpected answer, client does not support tis
mode
                              telnetState = 0;
                              uip_close();
                              connClosed();
                              return;
                        }
                  }
                  if( telnetState == rxExpectedCount ) {
                        // telnet setup completed
```

```
                        telnetState = 128; // state: idle
                        return;
                  }
                  switch( telnetState ) {
                        case 0:
                              txData.data[0] = TELNET_IAC;
                              txData.data[1] = TELNET_WILL;
                              txData.data[2] = TELNET_MODE_LINEMODE;
                              txData.length = 3;
                              uip_send( txData.data, txData.length );
                              break;
                        case 3:
                              txData.data[0] = TELNET_IAC;
                              txData.data[1] = TELNET_SB;
                              txData.data[2] = TELNET_MODE_LINEMODE;
                              txData.data[3] = TELNET_MODE_LINEMODE_EDIT;
                              txData.data[4] = 0;
                              txData.data[5] = TELNET_IAC;
                              txData.data[6] = TELNET_SE;
                              txData.length = 7;
                              uip_send( txData.data, txData.length );
                              break;
                        case 10:
                              txData.data[0] = TELNET_IAC;
                              txData.data[1] = TELNET_SB;
                              txData.data[2] = TELNET_MODE_LINEMODE;
                              txData.data[3] = TELNET_MODE_LINEMODE_TRAPSIG;
                              txData.data[4] = 0;
                              txData.data[5] = TELNET_IAC;
                              txData.data[6] = TELNET_SE;
                              txData.length = 7;
                              uip_send( txData.data, txData.length );
                              break;
                  }
            }
      }

      if( uip_rexmit() ) {
            uip_send( txData.data, txData.length );
      }
}
```