**STM Lesson 126. LAN8742A. LWIP. NETCONN. TCP. We connect two inspectors**

Posted on [July 31, 2018](http://narodstream.ru/stm-urok-126-lan8742a-lwip-netconn-tcp-soedinyaem-dva-kontrolera/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [LAN](http://narodstream.ru/lan/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-126-lan8742a-lwip-netconn-tcp-soedinyaem-dva-kontrolera/#respond)

[Learn the profession of Internet marketer!Free advertising **course** in Yandex. Profession in 10 days!Sign up before 10.09.18To learn moreprofiinternet.artur-grant.ru](https://an.yandex.ru/count/A-u3U3sF-Ka50E81CRt1ari00000ECgs7402I09Wl0Xe173qmTYm0e01Z9aAY06xoupnBP01_FobbYgO0QJyW_Wde07-_AMMAgW1dlU3-2Uu0PgfijKNm042s068r8uIu06wgvW6w04a-041Y083e0AixfmHkG9fZmrcEcyK3SW2mFUvlUUewQKoy0A7elqe-0A2W820WA0HW0FuZTH7Y0FJgPMd1vW3ceG1g0CIi0C4k0J_0UW4vm7u18xk0eW5Zku2a0M5gmEW1Ujjg0MANR05Ybsu1Sbim0MxW0p81T260z05yIdW1Gdm1G6O1e3GhFCEe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za6000jS740002f1_XHJ8MtsCL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW29QtpWY02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB46hDjsRuKG00qdxyLbjr1G302u2Z1SWBWDIJ0TaB-55CXRVOnKZe2uxk0l0B1eWCkA_UlW7e30AO3VYSPV8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ue_66xvRsuaW0?stat-id=3&test-tag=75866331701249&format-type=24&banner-test-tags=eyI1NjYwNTI0MzUwIjoiNzU4NjYzMDIzNDkzMTIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Buy robot constructorTeach your child to program. 6900 for a set of shares. OrderTo learn moredorado-uno.ru](https://an.yandex.ru/count/BZXAODjvhli50DS1CRx1ari00000ECgs7402I09Wl0Xe172AffoG1e01xCtW6OW1rBMagJMG0VZAnCelc07gZ-3LBg01zih4oY-e0UYEuDKkk07st_Rr6S010jW1ieMB5U01qBto4UW1OFW10OW21A02eBxP5ha2QOyDPZfl50t80f-mqOsbyToZLF02_hUyvGhu0eA0W820e1600_YDr4U80zEfbQS7i0C4w0JX0lW4wui2Y0NhYmAG1SA-0w05hL2e1On4i0MCHBW5md701R3x3CW5yAK3q0NI7k052V050PW6o8A66Q06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00BN1n0000gGVuKQpHzDZ5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YGtwmBW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n2ZKpR--540007I2rTRTGK0m0k0emN82u3Kam7P2_XHhD7qsCL8w0lhYmBm2mQ83ABwthu1w0m2c0t9icVo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AEXsRwUzk98?stat-id=4&test-tag=75866331701249&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjkyIjoiNzU4NjYzMDIzNDkzMTIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

We continue to work with the TCP protocol and today we will try to connect the server and client created by us in the previous sessions to each other and teach them to exchange data. I think this lesson can come in handy in the future, since it is often necessary to connect controllers among themselves, without resorting at all to participation in a network of computers.

Let's start with the server.

The project for the server will be created on the basis of the project of the [lesson 124](http://narodstream.ru/stm-urok-124-lan8742a-lwip-netconn-tcp-server/)**LAN8742\_TCP\_SERVER\_NETCONN** and give it a name a little different - **LAN8742\_TCP\_SERVER** , so that our projects from different classes will somehow differ in their names.

Let's open our project in Cube MX and, absolutely without touching it, we will generate the project for  **System Workbench**  and open it there. Set the optimization level to  **1** , remove it with debugging settings and comment out the lines unknown to the compiler in the **main.c** file  .

Let's try to build the project and start working with the **main.c** file  .

We will no longer send a string, but a 32-bit value, so in this connection, we will have to redo the code slightly.

Let's **go to the tcp\_thread** connection task function and declare there a variable for storing the number of passed system quanta from the moment the controller was turned on or rebooted, the same variable will serve for storing similar data that will come from the client

struct\_sock \*arg\_sock;

**uint32\_t syscnt = 0;**

In the same function, after we took the contents of the receive buffer into our local buffer, so as not to get confused, remove all the code to the end of the condition body. That is, there will only be this

if (recv\_err == *ERR\_OK*)

{

**netbuf\_data(inbuf, (void\*\*)&buf, &buflen);**

}

We'll finish the rest of the code now.

First, free the buffer memory, otherwise it will at one point stop receiving data

netbuf\_data(inbuf, (void\*\*)&buf, &buflen);

**netbuf\_delete(inbuf);**

If the buffer length is greater than 1, then we take the 32-bit value from the local buffer

netbuf\_delete(inbuf);

**if(buflen>1)**

**{**

**syscnt = \*(uint32\_t\*) buf;**

**}**

Display the value in the display using the queue

syscnt = \*(uint32\_t\*) buf;

**qstruct = osMailAlloc(strout\_Queue, osWaitForever);**

**qstruct->y\_pos = arg\_sock->y\_pos;**

**sprintf(qstruct->str,"%10lu",syscnt);**

**osMailPut(strout\_Queue, qstruct);**

We learn the number of passed system quanta and assign their value to the variable

osMailPut(strout\_Queue, qstruct);

**syscnt = osKernelSysTick();**

Send this quantity to the customer and wait a bit.

syscnt = osKernelSysTick();

**netconn\_write(newconn, (void \*) &syscnt, 4, NETCONN\_COPY);**

**osDelay(1);**

On the display a bit later, we will also display this value, and then free the memory of the queue

osDelay(1);

**qstruct->y\_pos = arg\_sock->y\_pos + 40;**

**sprintf(qstruct->str,"%10lu",syscnt);**

**osMailPut(strout\_Queue, qstruct);**

**osMailFree(strout\_Queue, qstruct);**

We will collect the code, we will issue the controller, because it seems that we have already finished the server.

Accordingly, we will not see anything except the cap, since we do not have a client yet.

Therefore, we will disconnect our motherboard from the computer and give it independent power, and with the computer we will connect the customer's card. You can also connect the cards to each other with a network cable.

The project will create for the client on the project ostnove [**last lesson**](http://narodstream.ru/stm-urok-125-lan8742a-lwip-netconn-tcp-client/)**LAN8742\_TCP\_CLIENT\_NETCONN** and call it simply **LAN8742\_TCP\_CLIENT** .

We will also open our project in Cube MX and, without touching it, we will generate a project for  **System Workbench**  and open it there. Set the optimization level to  **1** , remove it with debugging settings and comment out the lines unknown to the compiler for the video accelerator in the **main.c** file  .

Let's try to build the project and start working with the **main.c** file  .

In the default task function,  **StartDefaultTask,** correct the IP address of the server

IP4\_ADDR(&ServerIPaddr, **192, 168, 1, 191**);

The client port will also be different

err = netconn\_bind(conn, NULL, **1555**);

The server port will also change

err = netconn\_connect(conn, &ServerIPaddr, **80**);

Let's **move on to the send\_thread** send task function and delete this part of the code

~~if(syscnt>50000)~~

~~{~~

~~netconn\_close(conn);~~

~~netconn\_delete(conn);~~

~~qstruct = osMailAlloc(strout\_Queue, osWaitForever);~~

~~qstruct->y\_pos = 160;~~

~~strcpy(qstruct->str,"Connection was closed!");~~

~~osMailPut(strout\_Queue, qstruct);~~

~~osMailFree(strout\_Queue, qstruct);~~

~~osDelay(2);~~

~~break;~~

~~}~~

~~sprintf(buf,"%lu\r\n",syscnt);~~

We will not need a string buffer here either

uint32\_t syscnt = 0;

**~~char~~**

Let's correct the arguments in the call of the send function

sent\_err = netconn\_write(conn, **(void \*) &syscnt, 4,** NETCONN\_COPY);

Delete this

~~buf[strlen(buf)-2]=0;~~

~~strcpy(qstruct->str,buf);~~

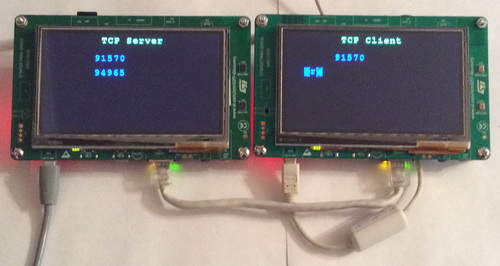
Instead, the line for the display is formed like this

qstruct->y\_pos = 60;

**sprintf(qstruct->str,"%10lu",syscnt);**

We can already check how we are transferring from the client to the server.

We will collect the code, we will sew the controller and see the indications on the displays of the nodes of our network. Perhaps after the firmware will have to reboot the server and the client



Excellent!

The data is transferred, and the data is already received by the client from the server, it is simply incorrectly displayed on the bottom line.

Now we will fix it.

Let's go into the function of the task of receiving packages  **recv\_thread** and declare there also a 32-bit variable

u16\_t buflen;

**uint32\_t syscnt = 0;**

In the body of the condition of sanity of the buffer value, we will take the data from the local buffer into our variable

if(buflen>1)

{

**syscnt = \*(uint32\_t\*) buf;**

Remove all this in this body

~~strncpy(str\_buf,(char\*)buf,buflen);~~

~~str\_buf[buflen-1]=0;~~

~~sprintf(qstruct->str,"%-20s", str\_buf);~~

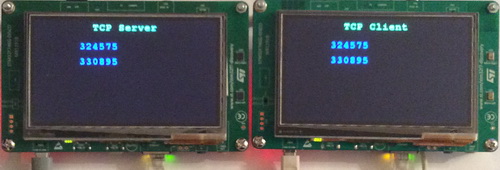
And instead we'll form a line for the display like this

qstruct->y\_pos = 100;

**sprintf(qstruct->str,"%10lu",syscnt);**

It seems to be everything.

We'll collect the code, restart the server, we'll run the client controller and see the result of the exchange on the displays of our connected cards



Everything is transmitted and accepted.

So, in this lesson we managed to connect two controllers over the network LAN using the NETCONN interface of the LWIP protocol stack, and also to organize data exchange between them.

Thank you all for attention!