**STM Lesson 95. LAN. W5500. FTP Server. Part 1**

Posted on [November 3, 2017](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-1/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-1/#respond)

[Development of an IS on SQL**Development of** programs on PostgreSQL, MySQL, MS **SQL** , SQLiteTo learn moreprojectdb.pro](https://an.yandex.ru/count/3GfEULSDQK050Cq1CSkuari00000ECgs7402I09Wl0Xe1736algX3O01lkQJh0680UFjfDi8a074gU6V0fW1w9h-c0Eu0QZxiU82m042s07eqAa3u07Yq8W9w07o0VW1OeW20w02tC3r4Ra2rbR1oEYxhutm0hUDfemF-0A2W820WB0IW0FPy-ly3eW3ey76dGgO0xJc2QW34h031EW4c0Ju1DFz1OW5q_q5a0NTk0YW1Tdb0QW5rC81i0NKmW6u1U6I0i05-zqHo0MEmGJG1SnNu0LQy0K1c0QGX8i9e0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XH7DBCjiD8i0U0W90Cq0S2u0U62lW70O080T08ceg0WS2GW0BW2AxW-WY02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB1geB47qeIBJuKG00vj61pbfr1G302u2Z1SWBWDIJ0TaB-54SqiosmqZe2zFz1V0B1eWCvexUlW7e30AO3PMCQF8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uey7wnDdjuKW0?stat-id=3&test-tag=80814134026241&format-type=24&banner-test-tags=eyIzMDE3Mzg1OTQiOiI4MDgxNDEwNDY0MTUzNiJ9&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Help Desk from 79 000 rubles a year!No restrictions on the number of users. Test for free!To learn morevsdesk.ru](https://an.yandex.ru/count/Oc_246n8df050Cy1CSwuari00000ECgs7402I09Wl0Xe173asxZP3O01tnY80OwZtgKqa07ky_A3BvW1pDYClYsu0S22jz4Pm042s076ZjeKu06KhP8Dw06G0lW1tg3UlW680WIW0k3SaH6v0jPMmSZekw-Dy0A-pPmO-0A2W820WB0IW0FgrSCzY0EnwPxR2vW3zRu5i0C4k0J_0UW4_07u19MV2OW5bPy9a0N6xmgW1VhY0gW5vwm2i0Ndh0Au1UJX0i05i84ao0NmamdG1Qw70U05Fl050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-57K3cctmqYm1u20a0pG1mBW1uOA-0S1W0W1q0YQYe21m9200k08iEx8280A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i8gWiGb22R7FXH0019uQZEMdK50C0BWAC5o0k0r9C1sGluKTGEQRV3IEWBbPy9y0i6Y0pUeDw-0UWC0fWD_BrcyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZWUE5xUtXI000?stat-id=4&test-tag=80814134026241&format-type=24&banner-test-tags=eyI2MzEyMzIwMjQ3IjoiODA4MTQxMDQ2NDE1MzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

After we learned how to organize several simultaneous connections with our **W5500** chip , you can think about the following protocol, which is intended for file transfer, and file system management over the network - the **FTP (File Transfer Protocol)** .

This protocol, although it has some problems with security, since the login and password for authorization here are transmitted in clear form, also has a popularity in our time. For example, at home, when sending data, we do not need to hide our credentials from ourselves, and we can also use the protocol for encryption on top of this protocol.

But along with the fact that the FTP protocol seems simple at first glance, it really is not so. The client with the server constantly exchanges messages, moreover, in order to transfer data, it is necessary to open one more connection in time, and, therefore, to close it in time. Of course, this protocol is simpler than, for example, protocols based on the NetBios protocol, which together with data transfer also provide support for encryption, naming names and other amenities. But, nevertheless, it will be necessary to tinker with it, which I actually did. There were many problems that met me on the way to mastering this protocol. For example, we have long learned how to transfer large amounts of data from our LAN modules, but we have not yet learned how to take it. It turned out to be not so simple. But, let's not be afraid of difficulties, otherwise we will not learn anything,

Let's talk a little about the protokle. Although we will not especially chew it, since then we still have to do it again when writing the algorithm of the program and writing the code that will implement this algorithm.

Therefore, we will get acquainted with the FTP protocol somehow in a nutshell.

As I have already said, the exchange between the client and the server in order to transfer some data then comes through messages. These messages are textual, however they start with a figure that carries the basic information.

Also, I repeat once again that for the full operation of the protocol we need two connections. One is control, and the other is for data transfer.

For the control connection over FTP, it was decided to use port **21** on the server . On the client side, the port is assigned automatically, that is, its number is not so important.

There are two types of port assignments for data transfer, since there are two modes of client and server operation for FTP - **active** and **passive** .

In the active FTP connection mode, the server initiates the connection. Then, to connect the data transfer to the server, port **20 is used** , and on the client side there is a port with a number larger than 1024. But in this case, if there is a firewall (NAT screen) or a NAT translator on the client's side, can not.

For this, there is also a passive mode. Then, the client initiates the connection to transfer the client's data to the server. In this case, both on the server side and on the client side, when creating a connection, random port numbers are assigned, only with a higher number than 1024. This connection is created as follows. The client tells the server that he wants to use passive mode, passing a certain command, and the server responds with the number of his IP address, as well as the port number with which the client will create a data connection.

Well, I propose the rest of the protoconality of the protocole when writing our source code.

To do this, we will create a project based on the project from **lesson 93 W5500\_HTTPS\_SOCKETS** call it **W5500\_FTPS** .

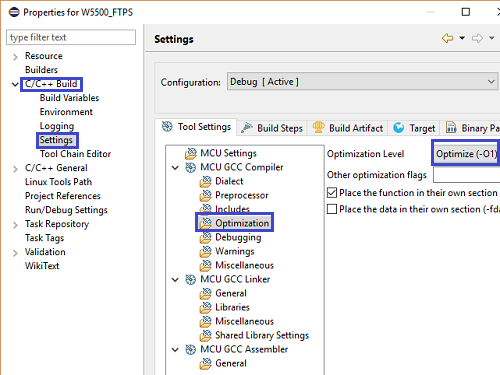
Run our project in the Cube MX.

We go into the project settings and select there as a programming environment for the **System Workbench** , since the file size of the firmware today risks to increase to a size of more than 32 kilobytes, which is inadmissible for the free version of Keil

Image00

Generate the project and connect it to the System Workbench environment.

Let's go into the project settings and set the optimization level **1**



Let's try to collect the project and start working with it.

Create a new library for FTP, consisting of two files - **ftpd.h** and **ftpd.c** with the following content

**ftpd.h** :

**#ifndef FTPD\_H\_**

**#define FTPD\_H\_**

**//--------------------------------------------------**

**#include "stm32f4xx\_hal.h"**

**#include <string.h>**

**#include <stdlib.h>**

**#include <stdint.h>**

**#include <ctype.h>**

**#include "fatfs.h"**

**#include "w5500.h"**

**//--------------------------------------------------**

**//--------------------------------------------------**

**#endif /\* FTPD\_H\_ \*/**

**ftpd.c** :

**#include "ftpd.h"**

**//-----------------------------------------------**

**extern UART\_HandleTypeDef huart2;**

**//-----------------------------------------------**

**extern char str1[60];**

**extern uint8\_t sect[515];**

**extern uint8\_t ipaddr[4];**

**extern volatile uint16\_t tcp\_size\_wnd;**

**//-----------------------------------------------**

In the header file, we connected a new file for us, **ctype.h** , which contains several functions for working not just with strings, but with separate string characters. This will come in handy later.

We also connect our library to **w5500.h** file below

#include "httpd.h"

**#include "ftpd.h"**

We will work with the protocol only in the passive mode. But we will add the port number for the active connection.

Therefore, we'll go into the file  **net.h** and add macros for the port numbers

#define LOCAL\_PORT 80

**#define LOCAL\_PORT\_FTP\_CONTROL 21**

**#define LOCAL\_PORT\_FTP\_DATA 20**

**#define LOCAL\_PORT\_FTP\_DATA\_PASSIV 35500**

And in the file **net.c** we add global variables, using these macros

uint16\_t local\_port = LOCAL\_PORT;

**uint16\_t local\_port\_ftp\_control = LOCAL\_PORT\_FTP\_CONTROL;**

**uint16\_t local\_port\_ftp\_data = LOCAL\_PORT\_FTP\_DATA;**

**uint16\_t local\_port\_ftp\_data\_passiv = LOCAL\_PORT\_FTP\_DATA\_PASSIV;**

Also we will connect these variables in the file **w5500.c**

extern uint16\_t local\_port;

**extern uint16\_t local\_port\_ftp\_control;**

**extern uint16\_t local\_port\_ftp\_data;**

**extern uint16\_t local\_port\_ftp\_data\_passiv;**

In the **w5500.h** file **we** add macros for sockets

#define MAC\_ADDR {0x00,0x15,0x42,0xBF,0xF0,0x51}

//--------------------------------------------------

**#define SOCKET\_MAX 8**

**#define FTP\_SOCKET\_MAX 2**

**#define FTP\_SOCKET\_CTRL 0**

**#define FTP\_SOCKET\_DATA 1**

//--------------------------------------------------

I think the purpose of macros is clear from their names.

In the file **net.c** in the function **packet\_receive,** change the parameter in the loop

for (i = **0** ; i < **SOCKET\_MAX** ; i ++)

In the function in the file w5500.c between the comment "// configure the sockets" and the delay will now be this code

//Настраиваем сокеты

**//Для FTP**

**//Открываем управляющий сокет**

**SetSockPort(0, local\_port\_ftp\_control);**

**OpenSocket(0,Mode\_TCP);**

**SocketInitWait(0);**

**//Начинаем слушать сокет**

**ListenSocket(0);**

**SocketListenWait(0);**

**//Открываем сокет для данных**

**SetSockPort(1, local\_port\_ftp\_data\_passiv);**

**OpenSocket(1,Mode\_TCP);**

**SocketInitWait(1);**

**//Начинаем слушать сокет**

**ListenSocket(1);**

**SocketListenWait(1);**

**//Для HTTP**

**for(i=FTP\_SOCKET\_MAX;i<SOCKET\_MAX;i++)**

**{**

**SetSockPort(i, local\_port);**

**//Открываем сокет**

**OpenSocket(i,Mode\_TCP);**

**SocketInitWait(i);**

**//Начинаем слушать сокет**

**ListenSocket(i);**

**SocketListenWait(i);**

**}**

HAL\_Delay(500);

That is, as you can see from the code, for the control connection FTP we use socket 0, for connection of FTP data transfer - socket 1, and the remaining 6 sockets remain for HTTP purposes. Therefore, we configured the sockets for FTP separately each due to the difference in their ports, and for HTTP sockets we use a loop.

Next, in the state view cycle, we will slightly adjust the parameters

for(i=**0**;i<**SOCKET\_MAX**;i++)

In the w5500.h file, we will create prototypes of some functions

void SendSocket(uint8\_t sock\_num);

**uint16\_t GetSizeRX(uint8\_t sock\_num);**

**void w5500\_readSockBuf(uint8\_t sock\_num, uint16\_t point, uint8\_t \*buf, uint16\_t len);**

Now, let's do some code improvement with respect to the HTTP protocol.

In the **httpd.c** file, **we** create a function at the bottom of the page to process HTTP packets, transferring to it almost all the packet processing code from the general function of receiving and processing packages of the file **w5500.c**

**//-----------------------------------------------**

**void http\_receive(uint8\_t sn)**

**{**

**uint16\_t point;**

**uint16\_t len;**

**if(httpsockprop[sn].data\_stat == DATA\_COMPLETED)**

**{**

**len = GetSizeRX(sn);**

**//Если пришел пустой пакет, то уходим из функции**

**if(!len) return;**

**//Отобразим размер принятых данных**

**sprintf(str1,"S%d len buf:0x%04Xrn",sn,len);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**//здесь обмениваемся информацией: на запрос документа от клиента отправляем ему запрошенный документ**

**//указатель на начало чтения приёмного буфера**

**point = GetReadPointer(sn);**

**w5500\_readSockBuf(sn, point, (uint8\_t\*)tmpbuf, 5);**

**if (strncmp(tmpbuf,"GET /", 5) == 0)**

**{**

**httpsockprop[sn].prt\_tp = PRT\_TCP\_HTTP;**

**http\_request(sn);**

**}**

**}**

**else if(httpsockprop[sn].data\_stat==DATA\_MIDDLE)**

**{**

**if(httpsockprop[sn].prt\_tp == PRT\_TCP\_HTTP)**

**{**

**tcp\_send\_http\_middle(sn);**

**}**

**}**

**else if(httpsockprop[sn].data\_stat==DATA\_LAST)**

**{**

**if(httpsockprop[sn].prt\_tp == PRT\_TCP\_HTTP)**

**{**

**tcp\_send\_http\_last(sn);**

**DisconnectSocket(sn); //Разъединяемся**

**SocketClosedWait(sn);**

**sprintf(str1,"S%d closedrn",sn);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**OpenSocket(sn,Mode\_TCP);**

**//Ждём инициализации сокета (статус SOCK\_INIT)**

**SocketInitWait(sn);**

**//Продолжаем слушать сокет**

**ListenSocket(sn);**

**SocketListenWait(sn);**

**}**

**}**

**}**

**//-----------------------------------------------**

We will also add a prototype to this function in the header file, and the prototype of the **http\_request**function  can now be removed from there.

After we now remove from the function  **w5500\_packetReceive** in the file **w5500.c** everything that we have transferred, it will take here such a short form

**//-----------------------------------------------**

**void w5500\_packetReceive(uint8\_t sn)**

**{**

**if(GetSocketStatus(sn)==SOCK\_ESTABLISHED)**

**{**

**//сокет для FTP**

**if(sn<FTP\_SOCKET\_MAX)**

**{**

**}**

**//сокет для HTTP**

**else if(sn>=FTP\_SOCKET\_MAX)**

**{**

**http\_receive(sn);**

**}**

**}**

**}**

**//-----------------------------------------------**

We'll collect the code and run the controller, in order to verify that we still have a normal HTTP server running by requesting a page from it from the browser.

Now let's start with FTP.

In the file **ftpd.c** we add the function of receiving and processing FTP packages, in which for the time being we only measure the size of the data in the package

**//-----------------------------------------------**

**void ftp\_receive(uint8\_t sn)**

**{**

**uint16\_t len;**

**uint16\_t point;**

**len = GetSizeRX(sn);**

**}**

**//-----------------------------------------------**

Create a prototype for this function and call it in the function  **w5500\_packetReceive** in the file **w5500.c**

if(sn<FTP\_SOCKET\_MAX)

{

**ftp\_receive(sn);**

}

Now, return to the **ftp\_receive** function  in  **ftpd.c** and write the following double condition

  len = GetSizeRX(sn);

**if(sn == FTP\_SOCKET\_CTRL)**

**{**

**}**

**else if(sn == FTP\_SOCKET\_DATA)**

**{**

**}**

}

In the header file **ftpd.h,** we will create an enumeration of possible FTP commands

#include "w5500.h"

**//--------------------------------------------------**

**// FTP commands**

**enum ftp\_cmd {**

***USER\_CMD*,**

***ACCT\_CMD*,**

***PASS\_CMD*,**

***TYPE\_CMD*,**

***LIST\_CMD*,**

***CWD\_CMD*,**

***DELE\_CMD*,**

***NAME\_CMD*,**

***QUIT\_CMD*,**

***RETR\_CMD*,**

***STOR\_CMD*,**

***PORT\_CMD*,**

***NLST\_CMD*,**

***PWD\_CMD*,**

***XPWD\_CMD*,**

***MKD\_CMD*,**

***XMKD\_CMD*,**

***XRMD\_CMD*,**

***RMD\_CMD*,**

***STRU\_CMD*,**

***MODE\_CMD*,**

***SYST\_CMD*,**

***XMD5\_CMD*,**

***XCWD\_CMD*,**

***FEAT\_CMD*,**

***PASV\_CMD*,**

***SIZE\_CMD*,**

***MLSD\_CMD*,**

***APPE\_CMD*,**

***NO\_CMD*,**

**};**

**//--------------------------------------------------**

I envied this enumeration from the example of WIZNET.

Also, below we add an FTP property structure

**//--------------------------------------------------**

**typedef struct ftp\_prop {**

**volatile uint8\_t connect\_stat;//статус состояния управляющего соеднинения FTP**

**volatile uint8\_t connect\_stat\_data;//статус состояния соеднинения для данных FTP**

**volatile uint8\_t login\_stat;//статус авторизации FTP**

**volatile uint8\_t ftp\_type;// Transfer type**

**volatile uint8\_t datasock\_mode;// режим соединения**

**volatile uint8\_t datasock\_state;// состояние соединения для данных**

**volatile uint8\_t data\_stat;//статус передачи данных**

**volatile uint16\_t rem\_bytes\_dirinfo;//кличество непереданных байтов информации о каталоге**

**volatile uint32\_t rem\_filesize;//количество непереданных байтов файла**

**enum ftp\_cmd current\_cmd;**

**char username[100];**

**char work\_dir[100];**

**char filename[100];**

**FIL my\_fil;**

**} ftp\_prop\_ptr;**

**//--------------------------------------------------**

Here, I hope, also the purpose of the fields is understandable. What could be incomprehensible, on the contrary, I wrote comments. Also, if something is not clear with this, then we will understand while using these fields.

After the structure is declared, we'll add some utility macros that we will use later

} ftp\_prop\_ptr;

**//--------------------------------------------------**

**#define FTP\_DISCONNECT 0**

**#define FTP\_CONNECT 1**

**//--------------------------------------------------**

**#define FTPS\_NOT\_LOGIN 0**

**#define FTPS\_LOGIN 1**

**//--------------------------------------------------**

**#define ASCII\_TYPE 0**

**#define IMAGE\_TYPE 1**

**#define LOGICAL\_TYPE 2**

**//--------------------------------------------------**

**#define DATASOCK\_IDLE 0**

**#define DATASOCK\_READY 1**

**#define DATASOCK\_START 2**

**//--------------------------------------------------**

**#define PASSIVE\_MODE 0**

**#define ACTIVE\_MODE 1**

**//--------------------------------------------------**

In the file **ftpd.c** we create a variable of the type of our structure

extern uint8\_t ipaddr[4];

**ftp\_prop\_ptr ftpprop;**

We return to the **ftp\_receive** function  and create a sector variable for writing to the buffer in the body of the condition

if(sn == FTP\_SOCKET\_CTRL)

{

**data\_sect\_ptr \*datasect = (void\*)sect;**

}

Also in the same body, we create another condition for checking for the fact of the created connection

data\_sect\_ptr \*datasect = (void\*)sect;

**if(!ftpprop.connect\_stat)**

**{**

**}**

**else**

**{**

**}**

In the main body (if the connection has not yet occurred), the current directory will be initialized. While this is the root directory

if(!ftpprop.connect\_stat)

{

**strcpy(ftpprop.work\_dir, "/");**

}

Let's **go** to the file **ftpd.h** and add there two macros - the server version number and the server name

#include "w5500.h"

//--------------------------------------------------

**#define HOSTNAME "w5500Chip"**

**#define VERSION "1.0"**

//--------------------------------------------------

Let's go back to the function  **ftp\_receive** file **ftpd.c** and in the same body conditions not connected to the buffer zanesom message with the name and version of the server that should answer our server to the client on a connection attempt to port 21

strcpy(ftpprop.work\_dir, "/");

**sprintf((char\*)datasect->data, "220 %s FTP version %s ready.rn", HOSTNAME, VERSION);**

Next, we must send this message to the server. For this, we do not yet have a function. Let's create it a little higher

extern volatile uint16\_t tcp\_size\_wnd;

**//-----------------------------------------------**

**void tcp\_send\_ftp\_one(uint8\_t sn, uint8\_t \* buf, uint16\_t len)**

**{**

**uint16\_t end\_point;**

**end\_point = GetWritePointer(sn);**

**end\_point+=len;**

**//Заполним данными буфер для отправки пакета**

**SetWritePointer(sn, end\_point);**

**end\_point = GetWritePointer(sn);**

**w5500\_writeSockBuf(sn, end\_point, buf, len);**

**//отправим данные**

**RecvSocket(sn);**

**SendSocket(sn);**

**}**

**//-----------------------------------------------**

This function is very similar to the function of sending a single-buffer HTTP packet, so there's nothing to tell, it's even much simpler. We here do not read anything from the map.

Let's return to the **ftp\_receive** function  in the same body where we just were, and send the message to the client, and then set the state "connected"

sprintf((char\*)datasect->data, "220 %s FTP version %s ready.rn", HOSTNAME, VERSION);

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)sect, strlen((char\*)datasect->data));**

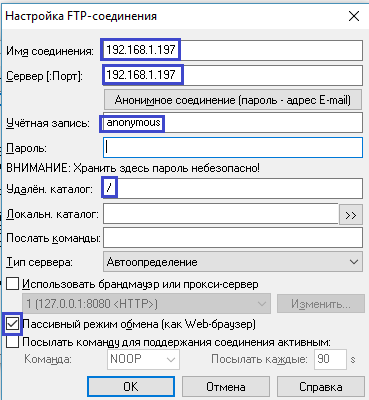
**ftpprop.connect\_stat = FTP\_CONNECT;**

Let's collect our code, we'll tell the controller and see if our message will return when we try to connect to our client's server, and see what command the client will send to us.

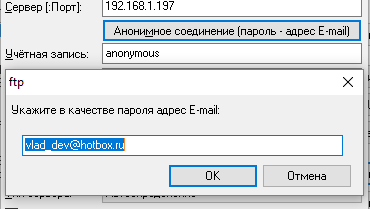
As an FTP server, I used the file manager Total Commander, which I have been using for a long time for various file operations on a PC. We will use the username and password for anonymous connection, there is a special login for this, and any email address is used as the password. Total Commander I have version **6.51**. It's quite an old version, but it suits me.

Also launch and WireShark.

In Total Commander, we call the menu item **FTP -> Connect to FTP server** , select "Add" in the opened dialog and fill in the following fields, without forgetting also the tick for passive mode, since we use this mode

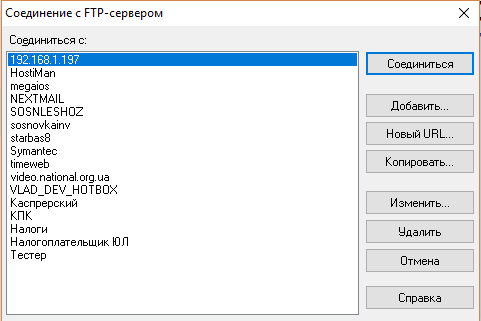


Also click on the button "Anonymous connection ..." and fill in some e-mail address (I filled out an address that has not existed for a long time)

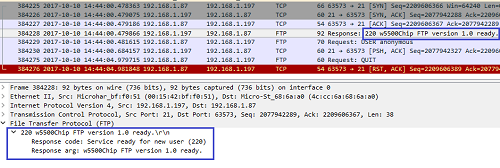


We agree, and in the dialogue with the credentials, too, we agree (click "OK").

We will have an FTP account. We press to connect, and then click "Abort", since we do not expect any connection, it is still far away. The process is important to us



We go to Wireshark and see what happens (we do not filter on FTP, otherwise we'll see only FTP packages, and we will not see the connection packages and other service packs) (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/10/Image05.png)

We see that our message about the server to the client came that it was recognized as FTP and the client sent us the USER command with the username, to which we also must now generate a response.

In the [**next part of the**](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-2/) lesson we will write the function of parsing the line of the FTP command, and we will also process several commands for an ad-hoc response to them to the client.

**STM Lesson 95. LAN. W5500. FTP Server. Part 2**

Posted on [November 4, 2017](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-2/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [4 comments ↓](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-2/#comments)

[Help Desk from 79 000 rubles a year!No restrictions on the number of users. Test for free!To learn morevsdesk.ru](https://an.yandex.ru/count/V7LQy6IGhYq50D81CKcxari00000ECgs7402I09Wl0Xe173asxZP3O01tnY80OwZtgKqa07ky_A3BvW1pDYClYsu0S22jz4Pm042s076ZjeKu06KhP8Dw06G0lW1tg3UlW680WEW0k3SaH6v0buGv5B-Imffy0AIWzoV2VW2We20W82G4u03wjN3FOW3iUcUsmkO0_M-1QW34h031BW4_m7e1FO1-0J_pWY81V_E2905slGEe0NpoWAe1RQO0h05jfW2k0MXvGF01R219CW5y9C9q0MOVk05Fl050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-54in_TEn4Ym1u20a0pG1mBW1uOA-0S1W0W1q0YQYe21m9200k08iEx8280A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGQEXKglXH000YZupqMdK50C0BWAC5o0k0r9C1sGluKIp7zqx4IEWB_yu8y0i6Y0pUeDw-0UWC0fWDbOneyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZiOJgtfFYI000?stat-id=3&test-tag=84112668909569&format-type=24&banner-test-tags=eyI2MzEyMzIwMjQ3IjoiODQxMTI2Mzk1MjQ4NjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

[Looking for Python courses?**Python programming** courses from scratch. Theory and practice. Start learning!To learn moreshultais.education](https://an.yandex.ru/count/Vsp82NOwFfe50E01CKgxari00000ECgs7402I09Wl0Xe173soUR12e01ZREU68W1iVwJcZAG0RBUk-ujc06-flosAw01zjkuxYse0Poc_BOhk07wyTk46S010jW1YCk-4E01tll01-W1fW7u0OIlthu1Y084e0AKz8qJkG9U4EHI_aiAQV02z9t8q0Vu0eA0W820a1E00-hwqiG7Y0Evn_MI3vW3thG8i0C4k0J_0UW4wmFu1Eo918W5x8a4a0Mau0IW1SUS0QW5reG1i0NMX06u1TEQ0S05uBCEo0NWr0FG1Taxu0K-y0K1c0RKkS3d0g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKSZ05q_4IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YA-iO9W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n2gWszD-5400AsDbFHQTGK0m0k0emN82u3Kam7P2_XHoC0NJyH8w0liYGJm2mQ838Ilthu1w0m2c0tylMRo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0EqQesi0u1eGy00000003mFwWF-AE1wExYa-98?stat-id=4&test-tag=84112668909569&format-type=24&banner-test-tags=eyI2MTU1NjMwNDg5IjoiODQxMTI2Mzk1NTc2MzIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of the**](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-1/) lesson, we set up the project, improved the code slightly in terms of working with the HTTP server, and also created the very first functions for working with the FTP protocol.

Back in the project in our function  **ftp\_receive** file **ftpd.c** in the same condition, but only in its opposite body (else). This body will be executed if the packet is received in the event that we already have a control connection. We add one more condition to this body, which will check that the packet came with data (not empty). Why should we respond to empty packets

  ftpprop.connect\_stat = FTP\_CONNECT;

}

else

{

**//проверим, что пакет не пустой**

**if(len > 0)**

**{**

**}**

}

Now go to the file **w5500.c**  and add there one more function of setting the pointer in the read buffer after the function **GetReadPointer**

**//-----------------------------------------------**

**void SetReadPointer(uint8\_t sock\_num, uint16\_t point)**

**{**

**uint8\_t opcode;**

**opcode = (((sock\_num<<2)|BSB\_S0)<<3)|OM\_FDM1;**

**w5500\_writeReg(opcode, Sn\_RX\_RD0, point>>8);**

**w5500\_writeReg(opcode, Sn\_RX\_RD1, (uint8\_t)point);**

**}**

**//-----------------------------------------------**

We add a prototype to this function in the header file.

Back in the file **ftpd.c** and fill the body of the condition we just created in the function **ftp\_receive**

if(len > 0)

{

**//Отобразим размер принятых данных**

**sprintf(str1,"S%d len buf:0x%04Xrn",sn,len);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**//примем данные**

**point = GetReadPointer(sn);**

**//Отобразим адрес данных**

**sprintf(str1,"S%d point RX:0x%04Xrn",sn,point);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**w5500\_readSockBuf(sn, point, sect, len);**

**//Передвинем указатель**

**SetReadPointer(sn, point+len);**

**RecvSocket(sn);**

**//завершим нулём**

**sect[len] = 0;**

**//отобразим их в терминальной программе**

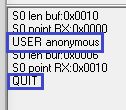
**sprintf(str1,"S%d: %srn", sn, (char\*)sect);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)(char\*)sect,strlen((char\*)sect),0x1000);**

}

The procedure for receiving packet data written in this body is standard, here more information is available in terminal programs. But for now let it be. We have to monitor the fact that we normally have the process of creating code. In this section of the code, we place the received data in the sect array and then display them in the terminal program.

We will collect the code, we will sew the controller and again we will try to connect to the FTP server, looking at the information in the terminal program



All commands are accepted and nominally displayed in the terminal program.

Now we need to somehow react to these commands.

We create above the function of parsing the data line of the package, creating immediately a series of variables, pointers and arrays

**//-----------------------------------------------**

**void ftp\_cmd\_parse(uint8\_t sn, char\* buf)**

**{**

**char \*\*cmd\_point, \*ch, \*arg, \*ch\_tmp;**

**char buf\_send[200];**

**uint16\_t len;**

**uint16\_t offset;**

**data\_sect\_ptr \*datasect = (void\*)buf\_send;**

**}**

**//-----------------------------------------------**

Call this function in the body of the condition at the end of the newly added code

      HAL\_UART\_Transmit(&huart2,(uint8\_t\*)(char\*)sect,strlen((char\*)sect),0x1000);

**ftp\_cmd\_parse(sn, (char\*)sect);**

    }

  }

}

else if(sn == FTP\_SOCKET\_DATA)

Create a global array of commands, represented in a lowercase form, as the FTP has no restrictions on this account and the commands can come in different registers. By the number of line items, the array must match the enumeration in the header file

extern volatile uint16\_t tcp\_size\_wnd;

//-----------------------------------------------

**/\* Command table \*/**

**static char \*ftp\_commands[] = {**

**"user",**

**"acct",**

**"pass",**

**"type",**

**"list",**

**"cwd",**

**"dele",**

**"name",**

**"quit",**

**"retr",**

**"stor",**

**"port",**

**"nlst",**

**"pwd",**

**"xpwd",**

**"mkd",**

**"xmkd",**

**"xrmd",**

**"rmd ",**

**"stru",**

**"mode",**

**"syst",**

**"xmd5",**

**"xcwd",**

**"feat",**

**"pasv",**

**"size",**

**"mlsd",**

**"appe",**

**NULL**

**};**

**//-----------------------------------------------**

Let's return now to the function **ftp\_cmd\_parse**  and begin to compose its body, and it promises to be considerable.

Convert the incoming command also to the lower case, converting the characters to the first blank or until the end of the data

data\_sect\_ptr \*datasect = (void\*)buf\_send;

**//преобразуем к нижнему регистру**

**for (ch = buf; \*ch != ' ' && \*ch != ''; ch++) \*ch = tolower(\*ch);**

Next, look for the team in our list

for (ch = buf; \*ch != ' ' && \*ch != ''; ch++) \*ch = tolower(\*ch);

**// найдём команду в списке**

**for (cmd\_point = ftp\_commands; \*cmd\_point != NULL; cmd\_point++)**

**{**

**if (strncmp(\*cmd\_point, buf, strlen(\*cmd\_point)) == 0) break;**

**}**

If we do not find it, we'll send in response that the command is non-existent and leave the function

  if (strncmp(\*cmd\_point, buf, strlen(\*cmd\_point)) == 0) break;

}

**if (\*cmd\_point == NULL)**

**{**

**sprintf((char\*)datasect->data, "500 Unknown command '%s'rn", buf);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, strlen((char\*)datasect->data));**

**return;**

**}**

Next, add a condition that knows whether the client is authorized on the server or not

  return;

}

**if (ftpprop.login\_stat == FTPS\_NOT\_LOGIN)**

**{**

**}**

Next, in the body of this condition, we add a switch condition that will recognize the options for the commands, in which the reaction to the three types of commands will be a way out of the condition, and if other commands come as known, since we have already filtered out the unknowns, that it is not yet ato- logized and will leave the function at all

if (ftpprop.login\_stat == FTPS\_NOT\_LOGIN)

{

**switch(cmd\_point - ftp\_commands)**

**{**

**case *USER\_CMD*:**

**case *PASS\_CMD*:**

**case *QUIT\_CMD*:**

**break;**

**default:**

**len = sprintf((char\*)datasect->data, "530 Please log in with USER and PASSrn");**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**return;**

**}**

}

Next, let's leave the unauthorized condition and set the pointer to the command argument

    return;

  }

}

**//установим указатель на аргумент команды клиента**

**arg = &buf[strlen(\*cmd\_point)];**

**while(\*arg == ' ') arg++;**

Then we begin to investigate which team came to us.

Add for this **switch**

while(\*arg == ' ') arg++;

**switch (cmd\_point - ftp\_commands)**

**{**

**}**

And now we'll start adding various command options to it.

At the moment, on the traffic analyzer, we saw that the client sent us the USER command with the username argument for processing. Here we will process it, at the same time add the default branch

switch (cmd\_point - ftp\_commands)

{

**case *USER\_CMD* :**

**break;**

**default: // Invalid**

**break;**

}

Let's start with default, so you do not forget.

default: // Invalid

**len =sprintf((char\*)datasect->data, "500 Unknown command '%s'rn", arg);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**HAL\_Delay(1000);**

**DisconnectSocket(sn); //Разъединяемся**

**SocketClosedWait(sn);**

**sprintf(str1,"S%d closedrn",sn);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**OpenSocket(sn,Mode\_TCP);**

**//Ждём инициализации сокета (статус SOCK\_INIT)**

**SocketInitWait(sn);**

**//Продолжаем слушать сокет**

**ListenSocket(sn);**

**SocketListenWait(sn);**

break;

Here we behave in a standard way: disconnect the socket, reopen it and start listening, displaying some information about it in the terminal program. The delay I turned on to avoid a very fast loop, I think it does not hurt much.

Now the actual processing of the **USER** command .

case *USER\_CMD* :

**sprintf(str1, "USER\_CMD : %s", arg);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**//отправим ответ на команду**

**len = strlen(arg);**

**arg[len - 1] = 0x00;**

**arg[len - 2] = 0x00;**

**strcpy(ftpprop.username, arg);**

**len = sprintf((char\*)datasect->data, "331 Enter PASS commandrn");**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

  break;

Here we show the command with an argument in the terminal program to see if we extracted it correctly, then we measure its length, zero the carriage return and go to a new line, copy the structure with properties into the field, then it will be the client's user name, then we send it the following message to the client requesting a password. Here we applied a new type of string length measurement while using the sprintf function. This function returns it as an outgoing argument.

Let's pass now to file  **w5500.c** and connect there a variable of our structure with properties

extern http\_sock\_prop\_ptr httpsockprop[8];

**extern ftp\_prop\_ptr ftpprop;**

Also I almost forgot. It is necessary in the function of receiving and processing packages **w5500\_packetReceive to**  handle the state of the disconnected socket. This is required in the event that the socket disconnects for some unknown reason, well, for example, at the initiative of the client. When we disable it intentionally, then we again create a connection with it. And there are other moments.

Therefore, we create a nasty case of the condition of an open socket in this function and write the corresponding code there

      http\_receive(sn);

    }

  }

**else**

**{**

**//управляющий сокет FTP**

**if(sn==FTP\_SOCKET\_CTRL)**

**{**

**if(ftpprop.connect\_stat) ftpprop.connect\_stat=FTP\_DISCONNECT;**

**if(ftpprop.login\_stat==FTPS\_LOGIN) ftpprop.login\_stat=FTPS\_NOT\_LOGIN;**

**SetReadPointer(FTP\_SOCKET\_CTRL, 0);**

**}**

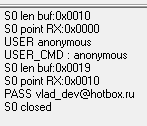
**}**

}

That is, in this case we reinitialize the settings and set the buffer pointer to 0.

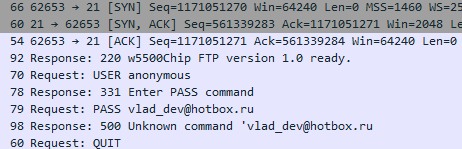
Let's now compile the code and run the controller and trace the response to the USER command from the server side, and also see what the client will answer us.

Here is the result in the terminal program



We see that the client responded by sending a password.

Also see what we are doing in the traffic analyzer



Here we also see that our client has received our message and answered us with his password, to which the server responded that it was for him an unknown team. Of course, it is not known to the server until we have processed it.

That's what we'll do. Back in the file **ftpd.c** and over the parsing function, add the client authorization function

**//-----------------------------------------------**

**void ftp\_login(uint8\_t sn, char \* pass)**

**{**

**char buf\_send[100];**

**uint16\_t len;**

**data\_sect\_ptr \*datasect = (void\*)buf\_send;**

**len = sprintf((char\*)datasect->data, "230 Logged onrn");**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**ftpprop.login\_stat = FTPS\_LOGIN;**

**}**

**//-----------------------------------------------**

In this function, we set the transfer data, send a message about the successful authorization of the client, and set the field of the structure with the authorization property to the authorized one. As such, we will not have authorization, we do not check the validity of the accounting pair, we simply agree with everything that the client will send us.

Now, return to our switch to the **ftp\_cmd\_parse** function  and process the **PASS** command there

  break;

**case *PASS\_CMD* :**

**len = strlen(arg);**

**arg[len - 1] = 0x00;**

**arg[len - 2] = 0x00;**

**ftp\_login(sn, arg);**

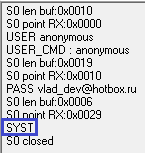
**break;**

default: // Invalid

It's all simple. We also reset the line feed and carriage return and call the client authorization function.

We will collect the code, we will tell the controller and see what this time the client will demand from us.

Result in the terminal program



The client sent us the **SYST** command , which is a request to identify the operating system, to which we need to tell him a little about our server.

We also look at the traffic analyzer, in which, of course, we will see the sending to the client of the fact that the given team is unknown to us (for now)

Image10

And the client turned out to be a very polite client and to our shameful answer about ignorance of the team did not take offense and requested the next team. But, I think, there are possible cases when an abusive client who just breaks the connection with us will fall, so let's still say something to him

  break;

**case *SYST\_CMD* :**

**len = sprintf((char\*)datasect->data, "215 UNIX emulated by W5500rnrn");**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**break;**

default: // Invalid

Let's check the reaction of the client, for which we will collect the code and let's edit the controller.

The reaction is the same - the client sent the **FEAT** command , which serves to reconcile the functions, that is, the client asks to tell the server now in more detail what it can.

We will answer him with our capabilities

  break;

**case *FEAT\_CMD* :**

**len = sprintf((char\*)datasect->data, "211-Features:rn MDTMrn REST STREAMrn SIZErn MLST size\*;type\*;create\*;modify\*;rn MLSDrn UTF8rn CLNTrn MFMTrn211 ENDrn");**

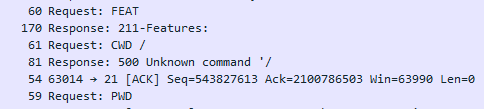
**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**break;**

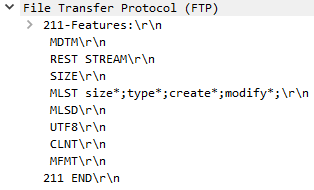
default: // Invalid

Here such here at us possibilities. Also there is a requirement that after enumerating all the possibilities it is necessary to send another word **END** , so that the client understands that before it was the last property of the server. What are these features, I will not list here, as it can be learned in special sources using the FTP protocol.

We will collect the project, we will impose the controller and check the result immediately in the Wireshark traffic analyzer, because there is more information there, in the terminal program we do not see the report about the packets coming from us



We see that the answer came from us. Let's see how it was deciphered by Wireshark



Here and so all beautifully saw the traffic analyzer.

The next command is sent to us by the CWD client, sees the server's response to the unknown, and then sends the Ping and PWD.

The first CWD command is the client's request to change the current directory. The name of the current directory the client passes in the argument. The client asks to go to the root directory, so as an argument, it passes the slash (slash).

Slowly start to answer him

  break;

**case *CWD\_CMD*:**

**len = strlen(arg);**

**arg[len - 1] = 0x00;**

**arg[len - 2] = 0x00;**

**//если не подкаталог**

**if(arg[0]=='/')**

**{**

**strcpy(ftpprop.work\_dir, arg);**

**}**

**break;**

default: // Invalid

Here we put the passed argument in the field with the name of the current directory (folder, directory). Only this if the client passed the root directory as an argument.

Next, we process the transfer by the client as an argument, go to the higher level. This is the case when the client asks from any directory that is currently out and go to the upper level. In this case, the client passes in the argument two points

  strcpy(ftpprop.work\_dir, arg);

}

**//переход на верхний уровень**

**else if(strncmp(arg,"..", 2) == 0)**

**{**

**}**

break;

Now we begin to write the body of this condition of transition to a higher level

else if(strncmp(arg,"..", 2) == 0)

{

**//отрежем каталог самого нижнего уровня**

**//Найдём последний символ '/'**

**offset = 0;**

**while(1)**

**{**

**ch\_tmp = strchr(ftpprop.work\_dir+offset,'/');**

**if(!ch\_tmp)**

**{**

**ftpprop.work\_dir[offset] = '';**

**break;**

**}**

**ch = ch\_tmp;**

**offset = ch\_tmp - ftpprop.work\_dir + 1;**

**}**

Here we use the name of the current directory of the structure stored in a certain field with the server properties. we will try to cut off the directory of the lowest level from it, so that the current directory becomes a higher-level directory. To do this, we look for the last character of the slash and, accordingly, we insert 0 into this place, which will correspond to the end of the line. Then we'll remove the slash, if the name of the directory ends, we do not need it. This does not apply to the root directory

    offset = ch\_tmp - ftpprop.work\_dir + 1;

  }

**//отрежем символ '/', если подкаталог**

**len = strlen(ftpprop.work\_dir);**

**if (len>1)**

**{**

**if(ftpprop.work\_dir[len-1]=='/')**

**{**

**ftpprop.work\_dir[len-1]=0;**

**}**

**}**

}

break;

Then we leave this condition (not from the cycle, but from the condition at all and we process all other conditions

      ftpprop.work\_dir[len-1]=0;

    }

  }

}

**else**

**{**

**//дополним путь символом '/' в конце, если его нет**

**len = strlen(ftpprop.work\_dir);**

**if(ftpprop.work\_dir[len-1]!='/') strcat(ftpprop.work\_dir,"/");**

**strcat(ftpprop.work\_dir, arg);**

**}**

break;

Here, on the contrary, we add the name to the slash.

Then we leave this body and give the client a response to the message

  strcat(ftpprop.work\_dir, arg);

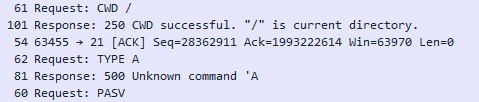
}

**len = sprintf((char\*)datasect->data, "250 CWD successful. "%s" is current directory.rn", ftpprop.work\_dir);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

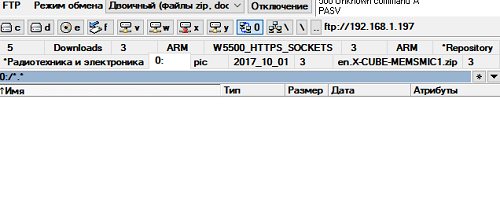
break;

We'll run the controller, collecting the pre-code and check that the server has processed the client command



The command was accepted by the client, to which the client responded by sending now not the PWD, but the other TYPE command. This is a request from the client that the server should work with the client in the ASCII standard. This is argued by the argument A. If, for example, the argument were B, then this would mean the client's request to apply a binary standard.

Also in Total Commander we already see that if we already have an FTP connection, but no list is currently displayed



This is because we have not yet transferred the list of files and directories of the current directory, but nobody asked for it yet. And this is done by the connection for data transmission. But before that we still get.

In the meantime, we'll process the client's command

  break;

**case *TYPE\_CMD* :**

**len = strlen(arg);**

**arg[len - 1] = 0x00;**

**arg[len - 2] = 0x00;**

**switch(arg[0])**

**{**

**case 'A':**

**case 'a': /\* Ascii \*/**

**ftpprop.ftp\_type = ASCII\_TYPE;**

**len = sprintf((char\*)datasect->data, "200 Type set to %srn", arg);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**break;**

**case 'B':**

**case 'b': /\* Binary \*/**

**case 'I':**

**case 'i': /\* Image \*/**

**ftpprop.ftp\_type = IMAGE\_TYPE;**

**len = sprintf((char\*)datasect->data, "200 Type set to %srn", arg);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**break;**

**default: /\* Invalid \*/**

**len = sprintf((char\*)datasect->data, "501 Unknown type "%s"rn", arg);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**break;**

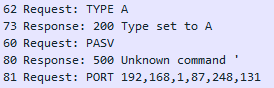
**}**

**break;**

default: // Invalid

Here at least the code and a lot, but to explain something, in fact there is nothing in it. Just react to the possible arguments and send relevant messages.

We will collect the code, we will tell the controller and see what the client will request further



We see that our client has received the message and now he sends us the **PASV** command . This command indicates the request of the client to use passive mode of exchange, since we set the corresponding checkbox in the FTP connection settings of the client. If the server agrees to this exchange mode, then it must respond with an IP address and a data connection port, since we already know that the connection will be separate for the data transfer. Since the command given for the server is still unknown, the server says that the client does not calm down and offers its own connection options.

In the [**next part of the**](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-3/) lesson, we will write the code for outputting information about the catalog to the client.

**STM Lesson 95. LAN. W5500. FTP Server. Part 3**

Posted on [November 5, 2017](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-3/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Published in [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-3/#respond)

[Help Desk from 79 000 rubles a year!No restrictions on the number of users. Test for free!To learn morevsdesk.ru](https://an.yandex.ru/count/2tXDYp5vmF050D81CMAxari00000ECgs7402I09Wl0Xe173asxZP3O01tnY80OwZtgKqa07ky_A3BvW1pDYClYsu0S22jz4Pm042s076ZjeKu06KhP8Dw06G0lW1tg3UlW680WEW0k3SaH6v0Yzvno1aV3Ely0Bsgjpn0VW2We20W82W5803wjN3FOW3iUcUsmkO0_M-1QW34h031BW4_m7e1EK1-0IybGY81RoL2905kVqDe0NIkGAe1UQ90h05vea2k0NvnWF01R219CW5y9C9q0NDT-05Fl050PW6WD2iymwW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0002WLW000Aa7-54iNDbKn4Ym1u20a0pG1mBW1uOA-0S1W0W1q0YQYe21m9200k08iEx8280A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGHSOwv_XH002Jz0JsMdK50C0BWAC5o0k0r9C1sGluKInSsLJ4IEWBl9K8y0i6Y0pUeDw-0UWC0fWDbOneyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3i24FR0E0Q4F00000000y3-e3_YZyRE8cvNYI000?stat-id=3&test-tag=89060471234561&format-type=24&banner-test-tags=eyI2MzEyMzIwMjQ3IjoiODkwNjA0NDE4NDk4NTYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

[Programming kitRobot, mobile application and study guide. Learn now is interesting!To learn moredorado-uno.ru](https://an.yandex.ru/count/0xToJuAyMKK50Dm1CMExari00000ECgs7402I09Wl0Xe172AffoG1e01d_tFcvogwEbHY06ZjgIfDP01Yih4oY-O0SIGuDKke06CoiJABwW1me_WrIwu0T3WzlKPm042s06oXOiLu07GlV8Hw05W-06Y-jw-0OW21A02eBxP5ha2Btd786HyCw_m0fsvhAK7-0A2W820WA0KW0EqyEs838W3n_FGpG-O0vlD0h031EW4YWJu182p0uW5WBC3a0N_uGEW1S220QW5p6wm1Snkk0N5Ui05iFiCo0NmfGFG1U8nu0LQy0K1c0QylEGPe0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000e5O0002f1_XHY8BvLCH8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW293Vh0k02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB49-4aohuKG00BFuCzbfr1G302u2Z1SWBWDIJ0TaB-568WlbKn4Ze2u2p0_0B1eWCelhUlW7e30AO3VozPl8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_uew7kZ9-LuaW0?stat-id=4&test-tag=89060471234561&format-type=24&banner-test-tags=eyI2Mzg2Mzg5NjM3IjoiODkwNjA0NDE4ODI2MjQifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of the**](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-2/) lesson we wrote the function of parsing the line of the FTP command, and we also processed several commands for an ad-hoc response to them to the client.

Let's now answer the client for the **PASV** command . Rather, we will start to answer, because it is not very simple and we will do this procedure gradually. To do this, we first connect the global variable

extern uint8\_t ipaddr[4];

**extern uint16\_t local\_port\_ftp\_data\_passiv;**

Then go to the **w5500.c** file and write the socket close function after the **SocketClosedWait** function

**//-----------------------------------------------**

**void CloseSocket(uint8\_t sock\_num)**

**{**

**uint8\_t opcode;**

**opcode = (((sock\_num<<2)|BSB\_S0)<<3)|OM\_FDM1;**

**w5500\_writeReg(opcode, Sn\_CR, 0x10); //CLOSE SOCKET**

**}**

**//-----------------------------------------------**

We add a prototype to this function, and at the same time some more in the header file **w5500.h**

void SetReadPointer(uint8\_t sock\_num, uint16\_t point);

**void CloseSocket(uint8\_t sock\_num);**

**uint8\_t GetSocketStatus(uint8\_t sock\_num);**

**void SetSockPort(uint8\_t sock\_num, uint16\_t port);**

Let us return to our switch to the function  **ftp\_cmd\_parse** file **ftpd.c** and begin to write the server response to the client's PASV command

  break;

**case *PASV\_CMD*:**

**local\_port\_ftp\_data\_passiv++;**

**if(local\_port\_ftp\_data\_passiv > 50000) local\_port\_ftp\_data\_passiv=35000;**

**break;**

default: // Invalid

Assign the server port address for the data connection. We will increment this value with each next connection, which will occur on the same client's command, so that the port addresses are different. If the port address reaches 50,000, the process will resume.

Next, we will respond to the client with an IP address and port address. First we pass the high byte of the port address, and then the younger one

if(local\_port\_ftp\_data\_passiv > 50000) local\_port\_ftp\_data\_passiv=35000;

**len = sprintf((char\*)datasect->data, "227 Entering Passive Mode (%d,%d,%d,%d,%d,%d)rn",**

**ipaddr[0], ipaddr[1], ipaddr[2], ipaddr[3], local\_port\_ftp\_data\_passiv >> 8, local\_port\_ftp\_data\_passiv & 0x00ff);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

break;

But it's not enough for us to simply transfer the team. We must ensure the operation of the data connection.

Therefore, we will do this

tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);

**//Разъединим сокет для данных и заново его подключим**

**if(GetSocketStatus(FTP\_SOCKET\_DATA)==SOCK\_ESTABLISHED)**

**DisconnectSocket(FTP\_SOCKET\_DATA); //Разъединяемся**

**CloseSocket(FTP\_SOCKET\_DATA);**

**SocketClosedWait(FTP\_SOCKET\_DATA);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)"654321rn",8,0x1000);**

**sprintf(str1,"S%d closedrn",FTP\_SOCKET\_DATA);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**SetSockPort(FTP\_SOCKET\_DATA, local\_port\_ftp\_data\_passiv);**

**OpenSocket(FTP\_SOCKET\_DATA,Mode\_TCP);**

**//Ждём инициализации сокета (статус SOCK\_INIT)**

**SocketInitWait(FTP\_SOCKET\_DATA);**

**//Продолжаем слушать сокет**

**ListenSocket(FTP\_SOCKET\_DATA);**

**SocketListenWait(FTP\_SOCKET\_DATA);**

**ftpprop.datasock\_mode = PASSIVE\_MODE;**

**ftpprop.datasock\_state = DATASOCK\_READY;**

**sprintf(str1,"PASV port: %drn", local\_port\_ftp\_data\_passiv);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

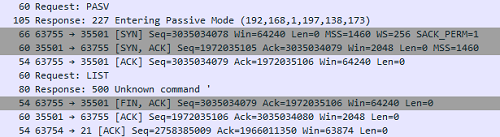
break;

Then I think everything is clear. If the connection is torn and reconnected, using the new port address.

With the processing of the answer like everything.

But now we need to somehow process the data packets in the data connection. But, in principle, we still have time to do this.

We will collect the code, we will sew the controller and once again we will try to connect



Here is the picture following. Our client received our answer, we had a connection, the client sends us the following LIST command, which means the client's request to send detailed information about the current directory. We do not yet know how to transmit such information, but it is transmitted by the connection already for data transmission. Therefore, the server sent a message to the client that he does not know such a command, to which the client asked to close the connection.

Therefore, the next task is to process the command from the **LIST** client .

But before we proceed with its implementation, let's also process the **QUIT** command , until we forget about it. This team, like many others, speaks for itself. This is the client's desire to disconnect from the server

  break;

**case *QUIT\_CMD* :**

**len = sprintf((char\*)datasect->data, "221 Goodbye!rn");**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**DisconnectSocket(sn); //Разъединяемся**

**SocketClosedWait(sn);**

**sprintf(str1,"S%d closedrn",sn);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**OpenSocket(sn,Mode\_TCP);**

**//Ждём инициализации сокета (статус SOCK\_INIT)**

**SocketInitWait(sn);**

**//Продолжаем слушать сокет**

**ListenSocket(sn);**

**SocketListenWait(sn);**

**ftpprop.connect\_stat = FTP\_DISCONNECT;**

**break;**

default: // Invalid

We send the right message to the client and disconnects it, then re-open a socket and listen to him, to be always ready to create a new connection on the client, not necessarily the same.

Now get to work with a team of **the LIST** , and along with it there is a similar team **MLSD** , which also can ask the server, so it is also processed

  break;

**case *MLSD\_CMD*:**

**len = sprintf((char\*)datasect->data, "150 Opening data channel for directory listing of "%s"rn", ftpprop.work\_dir);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**ftpprop.current\_cmd = *MLSD\_CMD*;**

**break;**

**case *LIST\_CMD*:**

**sprintf(str1,"LIST\_CMDrn");**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**len = sprintf((char\*)datasect->data, "150 Opening data channel for directory listing of "%s"rn", ftpprop.work\_dir);**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf\_send, len);**

**ftpprop.current\_cmd = *LIST\_CMD*;**

**break;**

case *QUIT\_CMD* :

Here we send relevant messages to the client. They are exactly the same. Also in the corresponding field of the structure we will enter the index of the current team.

Only now, after all this, we need to start sending information about the directory on the data connection. The transfer of this information is not an easy task and you can not form it right away. As there are certain formats of information about directories (name, type, date, size and other attributes).

But whether we want it or not, we must somehow provide this information. At the moment we have the current root directory. Yes it does not matter what our current catalog is. We could have requested anything from the client by writing the path to it in the connection properties.

Let's start working slowly with the information that we will transmit.

For now, let's move to the function ftp\_receive in the body of the condition of having a package in the data socket and in the event that we do not have the connection presence flag in the structure field, then we set it

else if(sn == FTP\_SOCKET\_DATA)

{

**if(!ftpprop.connect\_stat\_data)**

**{**

**sprintf(str1,"FTP Data socket Connectedrn");**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**ftpprop.connect\_stat\_data = 1;**

**}**

}

Then we add the determinant of the type of the current command, leaving only the body of one condition

      ftpprop.connect\_stat\_data = 1;

    }

**switch(ftpprop.current\_cmd)**

**{**

**case *LIST\_CMD*:**

**case *MLSD\_CMD*:**

**case *RETR\_CMD*:**

**if(ftpprop.data\_stat==DATA\_COMPLETED)**

**{**

**ftpprop.data\_stat=DATA\_FIRST;**

**}**

**break;**

**default:**

**break;**

**}**

  }

}

We processed several commands at once, because the handler is the same. In the handler, we learned the transmission status, and then, if we do not have non-transmitted data, set the data transfer status to the transmission status of the first packet. Unlike HTTP, here we will not differ, whether it is the first or the only one. This we will already determine as the package is processed. Also, in this condition, we will have a command to call a certain function, which we do not yet have. So let's add it. This will be the function of sending data to the client on the socket for data transfer. We add it after the function **ftp\_cmd\_parse**

**//-----------------------------------------------**

**void ftp\_data\_send(uint8\_t sn)**

**{**

**}**

**//-----------------------------------------------**

Now we will call it in our data packet handler in the data socket that we just wrote, in the same branch where we determined that we have not transmitted data.

  ftpprop.data\_stat=DATA\_FIRST;

**ftp\_data\_send(sn);**

}

break;

In the **w5500.h** header file **,** add a window size macro

#define SS\_DESELECT() HAL\_GPIO\_WritePin(CS\_GPIO\_PORT, CS\_PIN, GPIO\_PIN\_SET)

//--------------------------------------------------

**#define SIZE\_WND 2048**

Let us return to our client data transfer function  **ftp\_data\_send** file **ftpd.c** and begin writing her body

void ftp\_data\_send(uint8\_t sn)

{

**static char buf\_send[SIZE\_WND+3];**

**FRESULT res; //результат выполнения**

**buf\_send[0] = '';**

**switch(ftpprop.current\_cmd)**

**{**

**case *LIST\_CMD*:**

**case *MLSD\_CMD*:**

**break;**

**default:**

**break;**

**}**

}

While we will only handle the current commands request information in the directory.

Let's add several global variables for working with the FATFS library

extern volatile uint16\_t tcp\_size\_wnd;

**static FILINFO fileInfo;**

**static DIR dir;**

**extern uint32\_t bytesread;**

Let's return to our function and initialize some pointers in the handler for the information about the current directory

case *LIST\_CMD*:

case *MLSD\_CMD*:

**fileInfo.lfname = (char\*)sect;**

**fileInfo.lfsize = sizeof(sect);**

  break;

Above this function we will write one more separate function for transferring information about the catalog to the client, in which for now we will create a number of global variables and initialize the pointers.

**//-----------------------------------------------**

**//функция чтения и передачи позиций каталога**

**void ftp\_send\_dirinfo(uint8\_t sn, char \*buf)**

**{**

**FRESULT res; //результат выполнения**

**uint16\_t len;**

**char str\_date[15];**

**int str\_date\_ptr = 0, buf\_ptr = 0;**

**uint16\_t buf1\_ptr = 0;**

**char \*fn;**

**fileInfo.lfname = (char\*)sect;**

**fileInfo.lfsize = sizeof(sect);**

**}**

**//-----------------------------------------------**

This function will write information about the positions of the directory one position at the transfer buffer, and if suddenly we start to exceed the window size, the transfer will stop and be renewed already in the transmission of the next packet. We will also meet the situation almost always when we reach the maximum buffer size, but the current position will not be transmitted to the end. Therefore, we will have to memorize the position of the pointer in the line with information about the current position and return to this place in the next packet transfer. So create another global buffer

extern uint8\_t sect[515];

**static char buf1[256];**

I think this size is enough for information about the position. Also, in order to ensure that our buffer does not fall apart to our next arrival in the function, we add to it the attribute **static** .

Let's continue to write the body of our function

fileInfo.lfsize = sizeof(sect);

**buf\_ptr+=3;//оставим место под ардес и опкод**

**//если есть оставшаяся информация о позиции в каталоге, то передадим её**

**if(ftpprop.rem\_bytes\_dirinfo)**

**{**

**memcpy((uint8\_t\*)(buf+buf\_ptr),(uint8\_t\*)(buf1+strlen(buf1)-ftpprop.rem\_bytes\_dirinfo),ftpprop.rem\_bytes\_dirinfo);**

**buf\_ptr+=ftpprop.rem\_bytes\_dirinfo;**

**}**

We left the space under the address and opcode, then, if we have unmapped bytes from the previous position, we will add them to the main buffer, which we will then transfer.

Next cycle

  buf\_ptr+=ftpprop.rem\_bytes\_dirinfo;

}

**while(1)**

**{**

**buf1\_ptr = 0;**

**res = f\_readdir(&dir, &fileInfo);**

**}**

In the loop, we set the buffer pointer for the current position to zero and read the information about the current position. We worked with this function and know perfectly well that it itself switches the pointer to the next position after it is recalled again.

We work further with the cycle

res = f\_readdir(&dir, &fileInfo);

**if (res==*FR\_OK* && fileInfo.fname[0])**

**{**

**fn = fileInfo.lfname;**

**if(!strlen(fn)) fn=fileInfo.fname;**

**}**

**else**

**{**

**break;**

**}**

We first determine the name of the file (directory). If it is long, it is already defined somewhat differently, we also know how this is done, so we have also taken this into account here.

Do not exit the loop and write the code further

  break;

}

**switch((fileInfo.fdate >> 5) & 0x0f)**

**{**

**case 1:**

**len = sprintf(str\_date, "JAN ");**

**break;**

**case 2:**

**len = sprintf(str\_date, "FEB ");**

**break;**

**case 3:**

**len = sprintf(str\_date, "MAR ");**

**break;**

**case 4:**

**len = sprintf(str\_date, "APR ");**

**break;**

**case 5:**

**len = sprintf(str\_date, "MAY ");**

**break;**

**case 6:**

**len = sprintf(str\_date, "JUN ");**

**break;**

**case 7:**

**len = sprintf(str\_date, "JUL ");**

**break;**

**case 8:**

**len = sprintf(str\_date, "AUG ");**

**break;**

**case 9:**

**len = sprintf(str\_date, "SEP ");**

**break;**

**case 10:**

**len = sprintf(str\_date, "OCT ");**

**break;**

**case 11:**

**len = sprintf(str\_date, "NOV ");**

**break;**

**case 12:**

**len = sprintf(str\_date, "DEC ");**

**break;**

**}**

This section of code, I think, is understandable. We take information from the structure of the information about the file (catalog) about the month of creating the file (catalog) and put it in a line in a format understandable to the client.

We write further the code in the loop without leaving it

  break;

}

**str\_date\_ptr += len;**

**len = sprintf(str\_date + str\_date\_ptr, "%d ", (fileInfo.fdate & 0x1f));**

**str\_date\_ptr += len;**

**len = sprintf(str\_date + str\_date\_ptr, "%d", (((fileInfo.fdate >> 9) & 0x7f) + 1980));**

**str\_date\_ptr = 0;**

**if(fileInfo.fattrib & AM\_DIR)**

**{**

**sprintf(buf1, "d");**

**}**

**else**

**{**

**sprintf(buf1, "-");**

**}**

Here we move on the buffer, intended solely for the string representation of the date, then we enter the date of the month into the given buffer. Next, we reset the buffer counter, and then we begin to fill the buffer of the position itself. We start to fill it with information about the type of position - this is a file or a directory.

We write further the body of our cycle

  sprintf(buf1, "-");

}

**buf1\_ptr++;**

**len = sprintf(buf1 + buf1\_ptr, "rwxr-xr-x 1 ftp ftp %lu %s %srn", fileInfo.fsize, str\_date, fn);**

**buf1\_ptr += len;**

We move along the buffer one position and begin to fill it according to the standard, defining the client's rights to the file. We will give the standard rights, then we will pass the string "ftp ftp", which is also mandatory, and then the information about the file size and zarenee prepared by us earlier the string with a string representation of the date. Well, move the pointer to the size of the data transferred to the buffer.

Continue the cycle body further

buf1\_ptr += len;

**if((buf\_ptr+buf1\_ptr)>(SIZE\_WND+3))**

**{**

**memcpy((uint8\_t\*)(buf+buf\_ptr),(uint8\_t\*)buf1,SIZE\_WND+3-buf\_ptr);**

**}**

**else**

**{**

**memcpy((uint8\_t\*)(buf+buf\_ptr),(uint8\_t\*)buf1,buf1\_ptr);**

**}**

**buf\_ptr += buf1\_ptr;**

Accordingly, here we already copy the buffer with the line for the position to the large buffer, which we are preparing to write to the buffer for sending to the client, equal to the size of the window. And then we just determine whether we have reached the maximum size of the main socket buffer. If you have reached, then copy it into only that part of the position that will climb into it as they say to the eyeballs. And if they do not, then copy everything. Then again we move around the main buffer.

Then we write our cycle. This will be the final code of our cycle

  buf\_ptr += buf1\_ptr;

**if(buf\_ptr>(SIZE\_WND+3))**

**{**

**ftpprop.rem\_bytes\_dirinfo = buf\_ptr-SIZE\_WND-3;**

**ftpprop.data\_stat=DATA\_MIDDLE;**

**break;**

**}**

**else**

**{**

**ftpprop.rem\_bytes\_dirinfo = 0;**

**}**

}

Here, when the end of the buffer is reached, we determine and set the value of the remaining byte field for the current position transmission, and also change the transfer status to the transfer status of the middle part of the entire data volume. Until we know whether it will be really average or will be the last. Also do not forget about the three information bytes. We will define this later. Well, if you do not reach the end of the buffer, then reset the counter.

Now we leave the infinite loop and write the code further. We now have to send our filled buffer to the client

      ftpprop.rem\_bytes\_dirinfo = 0;

    }

  }

**if(buf\_ptr>(SIZE\_WND+3))**

**{**

**buf\_ptr=SIZE\_WND+3;**

**}**

**else**

**{**

**ftpprop.data\_stat=DATA\_END;**

**f\_closedir(&dir);**

**}**

**tcp\_send\_ftp\_one(sn, (uint8\_t \*)buf, buf\_ptr-3);//отнимаем из общей длины служебные байты**

}

Here we move the pointer in the buffer to its very end when the maximum size is reached. Otherwise we do not touch its value, since it is not exceeded and we close the directory. Also, otherwise, we set the status of the data transfer to the end. And then we pass the buffer to the client.

We will process all these statuses later, but for now return to our data transfer function to the client **ftp\_data\_send** and continue to write there the client's command processor for requesting the contents of the directory

fileInfo.lfsize = sizeof(sect);

**if(ftpprop.data\_stat==DATA\_FIRST)**

**{**

**res = f\_opendir(&dir, ftpprop.work\_dir);**

**if(res == *FR\_OK*)**

**{**

**ftp\_send\_dirinfo(sn,buf\_send);**

**if(ftpprop.data\_stat==DATA\_END)**

**{**

**f\_closedir(&dir);**

**}**

**}**

**}**

**else if(ftpprop.data\_stat==DATA\_MIDDLE)**

**{**

**ftp\_send\_dirinfo(sn,buf\_send);**

**}**

break;

Here, provided that we have this first package, we open the directory. If the directory was successfully opened, then we call our function to transfer information about the contents of the directory and then, if we have the status of the end of the data transfer, close the directory. Otherwise, if we have the status of data transfer of the middle package, we do not open the directory anymore, it means that we have already opened it earlier and continue to transmit data with information about the next positions of the directory, for which we will call the corresponding function. While with this data transfer function everything.

Let's return now to the function of receiving and processing **ftp\_receive** FTP **packets** and in the body of the condition of the packet receipt for the data connection above the switch of the current command, we add one more switch of the data transfer statuses and process two statuses in it

  ftpprop.connect\_stat\_data = 1;

}

**switch(ftpprop.data\_stat)**

**{**

**case DATA\_END:**

**ftpprop.current\_cmd = *NO\_CMD*;**

**len = sprintf((char \*)(sect+3), "226 Successfully transferred "%s"rn", ftpprop.work\_dir);**

**tcp\_send\_ftp\_one(FTP\_SOCKET\_CTRL, sect, len);**

**ftpprop.data\_stat=DATA\_COMPLETED;**

**DisconnectSocket(FTP\_SOCKET\_DATA); //Разъединяемся**

**SocketClosedWait(FTP\_SOCKET\_DATA);**

**sprintf(str1,"S%d closedrn",FTP\_SOCKET\_DATA);**

**HAL\_UART\_Transmit(&huart2,(uint8\_t\*)str1,strlen(str1),0x1000);**

**OpenSocket(FTP\_SOCKET\_DATA,Mode\_TCP);**

**//Ждём инициализации сокета (статус SOCK\_INIT)**

**SocketInitWait(FTP\_SOCKET\_DATA);**

**//Продолжаем слушать сокет**

**ListenSocket(FTP\_SOCKET\_DATA);**

**SocketListenWait(FTP\_SOCKET\_DATA);**

**ftpprop.connect\_stat\_data = 0;**

**break;**

**case DATA\_MIDDLE:**

**ftp\_data\_send(sn);**

**break;**

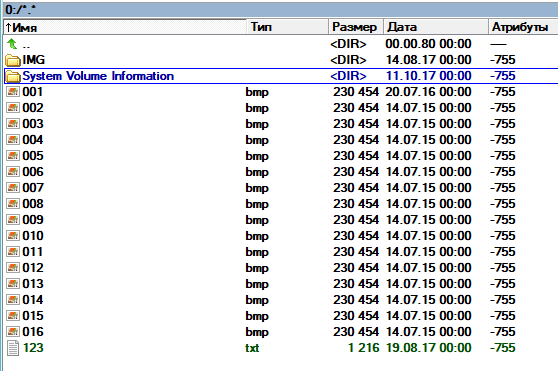
**}**

switch(ftpprop.current\_cmd)

In the case of the end of transmission status, we close the socket, re-initialize certain fields of the structure, send a specific message to the client that the data has been transferred so that the client knows that we have transferred everything to him, open the socket and start listening to it again. And in the case of the transfer status of the middle part, we call the data transfer function again.

We will collect the code, we will sew the controller. First, fill the flash card with the calculation that the information about the directory does not exceed the size of the buffer.

Connect to the server and see the result



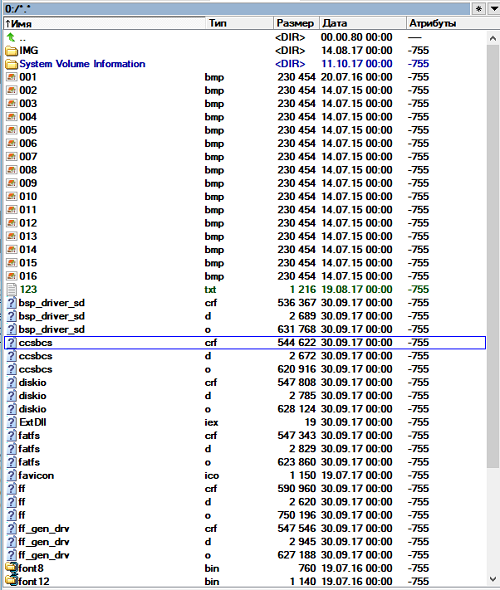
Here is the result. We see the entire file system of the flash card.

But the information from WireShark

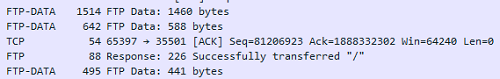
Image18

We see that everything has been transferred.

Now fill the map with more information and see the result

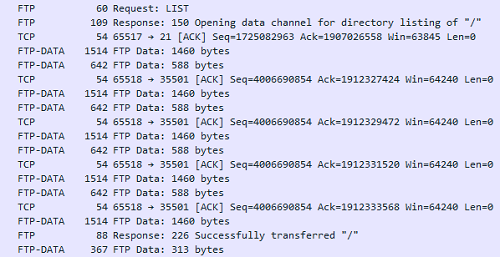


Also all information is displayed and comes. Here is the result in WireShark, where we see the sending of the next packet that did not fit into the buffer

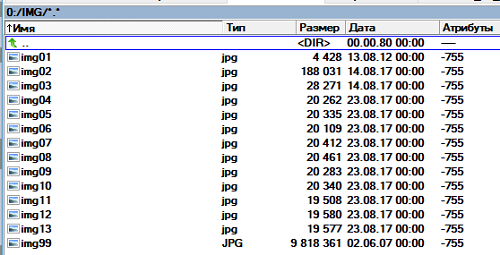


We will fill even more our storage with information.

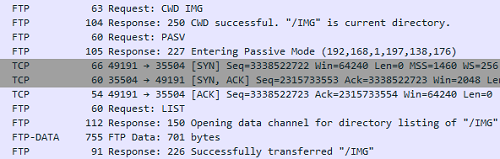
And here is the result



We work further with FTP. Let's try to open a catalog with a double click of the mouse

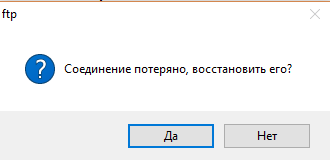


The catalog opened normally. Let's see the information in the traffic analyzer



Here, too, everything is fine. The client and the server communicate normally and adequately respond to messages and commands.

Now let's go further. Let's try to return from our catalog to the level above, that is, to get out of it



In this case, the connection is lost. The reason for this gap is found in the [**next part of**](http://narodstream.ru/stm-urok-95-lan-w5500-ftp-server-chast-4/) our lesson, in which we will also learn how to create and delete files and directories, and also transfer files to the server and from the server.