**STM Lesson 128. LAN8742A. LWIP. NETCONN. HTTP. AJAX. Part 1**

Posted on [August 21, 2018](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-1/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [FreeRTOS](http://narodstream.ru/freertos/) , [LAN](http://narodstream.ru/lan/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-1/#respond)

[PHP courses from scratch2 online lectures a week, personal mentor, final project site, certificate.To learn morehtmlacademy.ru](https://an.yandex.ru/count/LXkhlYDJRmC50Du1CHR2ari00000ECgs7402I09Wl0Xe173MZO-53e01WuA90uW1s9sQr3IG0QR6ww0lc07gk-xvBQ01hiRhe2-e0PAxxlajk07edVpU6S010jW1YhNh5E01rBs61-W1fWBu0SBQtxu1Y083e0AinQmFkGBInzxJWSPEj_02i_Jodmdu0eA0W820i1600xEwYSuEY0FFxzhU2gW34h031BW4_m7e18W2-0Id-0M81QVu1P05YiS8e0MCv06e1QR10R05fi41k0MqbWB01Pkq6iW5iDi6q0NzLk05Fl050PW6WBY6_WsW1ku2g0Rk0ia6nYkJMac0THFH1iQ_oD59W7KJsGO0001mSG000Aa7-54uJFpjnKYm1u20a0pG1mBW1uOA-0S1W0W1q0YwYe21m9200k08rgBM1u0A0S4A00000000y3_O2WBW2e29UlWAmFmLY0i6gWiGBrSbjlXH001ri4rSMtK50C0BWAC5o0k0r9C1sGluKJXC_Et5IEWBf_W5y0i6Y0p2sj--0UWC0fWD-9nbyWq0-Wq0WWu0YGu00000003mFv0Em8Gzc0x6rypKbOdWkXEW3gAa3R0E0Q4F00000000y3-e3_YZqRMul_lYI000?stat-id=3&test-tag=76417161256961&format-type=24&banner-test-tags=eyI2MzQyNjY4NjkxIjoiNzY0MTYwNTgxNjMyMDAifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

[Looking for Python courses?**Python** programming **courses** from scratch. Theory and practice. Start learning!](https://an.yandex.ru/count/7fR9C_ItcBu50Dy1CHV2ari00000ECgs7402I09Wl0Xe173soUR12e01YltApWA80RZ-aveoa06othlkBPW1lgRyjYkW0VRRkEujg06SflosAxW1-l7RX1d00GBO0OZBlX3W0Txxm0Ve0QO1-064hzw-0OW21A02bFID4xa2qiVUqu76JhVm0ldatk42-0A2W820WB0HW0FChul528W3repMxWQO0wdM2B031BW4_m7e1A83-0IRmWE81Pl20v05-S84e0M8Xm6e1Rjoi0MxShW5c9C1m0NWimx81U3K0z05m3FW1Lhm1G6O1kAAs-K2e0Rk0gW6xWB91iOharf9W7KJqGR6lyZHIO1r4za60000S740002f1_XHj4KgxiL8i0U0W90Cq0S2u0U62lW70O080T08keg0WS2GW0BW28hwnWc02W712W0000000F0_s0e2u0g0YNhu2i3y5OWB2AeB4A4ZlPluKG00u2vPN5jr1G302u2Z1SWBWDIJ0TaB-56qHIhknKZe2vl20_0B1eWCXA_UlW7e30AO3ScoP_8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0wWtJQm3W6X3m0000000F0_g0_ueu76biNxuaW0?stat-id=4&test-tag=76417161256961&format-type=24&banner-test-tags=eyI2MTU1NjMwNDg5IjoiNzY0MTYwNTgxNjMyMDAifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

We continue to work with **LWIP NETCONN** , as well as with the **HTTP protocol** .

And today we will try not only to send a page to the browser from our controller, but to send data to a certain place on the page, without rebooting the whole page.

Previously, we could not do this, because we used CGI and SSI technologies, which could not afford this.

And that's why AJAX technology came to our aid, which makes it possible.

**AJAX** stands for **Asynchronous Javascript and XML**. From this decoding it follows that AJAX uses the JavaScript language. I hope we all know that this language is supported by all browsers. This language is not for the server, especially not for our controller. Therefore, we will write code for AJAX in the page or in a separate file with the script, and not in the code for the controller. But nevertheless, we will also have a task for the controller - to provide the data that the browser will ask us for using AJAX technology, and also to accept the data that the browser will give us, which is also possible. Also from the decoding of the abbreviation, we see that AJAX is the ability to exchange information between the browser and the server in asynchronous mode, that is, we requested data from the server from the browser, and the server has not yet been transferred to them, it still prepares them, but their browser does not wait, but continues to execute further code, and when the data comes, the browser will process it. This ensures that the page output does not slow down due to waiting for data from the server. Therefore, AJAX technology allows us not only to exchange browser data with the server without reloading the entire page, and also, if this data will be very large, it also does not cause a slowdown in the loading of the page content.

I, of course, will not here lay out all the subtleties of the JavaScript language and CSS styling. We just use them to pursue our goals. Who is interested in learning how to fully operate with these languages, then there is a lot of lessons and information on this subject, though not with me. I could certainly give good lessons on these languages ​​and not only, but, alas, everything is impossible to catch.

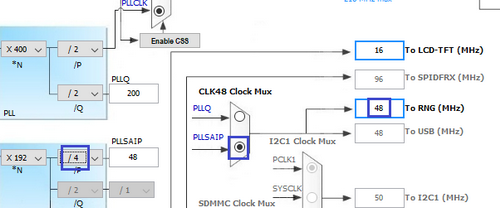
So let's move on to the project for now.

The project was made from the project of the [**last lesson**](http://narodstream.ru/stm-urok-127-lan8742a-lwip-netconn-http-server/) **LAN8742\_HTTP\_SERVER\_NETCONN** and was named **LAN8742\_HTTP\_SERVER\_AJAX\_NETCONN** due to the use of AJAX technology.

Open our project in the Cube MX and turn on the random variable generator **RNG**

http://narodstream.ru/wp-content/uploads/2018/08/stm128img11.png

Then go to Clock Configuration and change the controls a bit



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 .

While the project we will not collect, as we know. that he still does not meet.

Let's start while preparing the page file.

Create our page named **index.html** , save it in the folder fs of the directory where we **compile the** file **fsdata.c** , and open it in any editor. It is more convenient, of course, to do this, for example in **SublimeText** , but if it is not near at hand, notepad **++** notepad will do .

The original text of our page will be as follows:

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <!DOCTYPE html>  <html lang="en">    <head>      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />    </head>    <body>    <div id="page">      <div id="header">      </div>      <div id="content">        <div id="article">        </div>        <div id="sidebar">        </div>      </div>      <div id="footer">      </div>    </div>    </body>  </html> |

This is a normal WEB-document with a header and a breakdown into blocks. We will have a block with a title, a block with content and a block with a basement. The block with content will be divided into 2 parts - the main content with the "article" ID, as well as the sidebar (the right side panel) with the identifier "sidebar". If we try to open such a page in the browser, then we will not see anything. Our blocks will follow one another, they will be wide in the whole page and with a height of 0, since there is no content in them. But we will correct it due to stylization, it's not for nothing that we have assigned identifiers to each block. For this we need to create a script with the styles of our blocks, we can do this in the text of the page itself, but in order not to load it with extra information for better readability, we will put our block styles and then not only the blocks in a separate file,**style.css** and put it in the folder with the page itself.

The initial contents of this file will be the following

CSS



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | body {    background-color: #333333;    background-image: url(IMG/bg01.png);  }  #page {    width: 960px;    margin: 0 auto;  }  #header {    background-image: url(IMG/bg02.jpg);    background-blend-mode: ;  height: 100px;  }  #content::before, #content::after {    content: "";    display: table;  }  #content::after {    clear: both;  }  #content {    background-image: url(IMG/bg03.png);    zoom: 1;  }  #article {    background-color: rgba(255,255,255,.7);    width: 720px;    padding: 20px;    float: left;  }  #sidebar {    width: 200px;    float: left;  }  #sidebar ul li {    color: #8f8;  }  #footer {    background-color: #330033;    height: 100px;    padding: 20px;  }  #footer p {    color: #0ff;  } |

We set a certain type of location of our blocks, also applied the background to them, in some places even in the form of pictures, gave them a certain transparency. Who knows css, he will understand.

Also we do not forget to put our file from the last lesson **404.html** in the folder with pages to give it to the user when requesting a nonexistent document.

Also put the **IMG** folder with the image files for backgrounds in the folder with our page.

In addition to these pictures, we put four more pictures from the last lesson there img01.jpg - img04.jpg.

Also, do not forget to connect the script file in our **index.html** page in the **head** tags

XHTML



|  |  |
| --- | --- |
| 1  2  3 | <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />    <link rel="stylesheet" href="style.css">  </head> |

And, even if we open our page in the browser now, we will not see anything there either, since we have empty blocks.

Therefore, we will start to slowly fill them with contents. The content we will take from the page of the last lesson.

Block **header** :

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <div id="header">    <h1 style="text-align: center; color: #fff;">STM32F746G-DISCO</h1>    <p></p>    <p></p>    <h2 style="text-align: center; color: #fff;">User Manual</h2>  </div> |

Block **article** :

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <div id="article">    <h1 style="text-align: center;">      <strong>Introduction</strong>    </h1>    <p style="text-align: justify; margin-left: 11.35pt;">      The STM32F746G-DISCO discovery board (32F746GDISCOVERY) is a complete demonstration and development platform for STMicroelectronics ARM<sup>&reg;</sup>     Cortex<sup>&reg;</sup>-M7 corebased STM32F746NGH6 microcontroller. This microcontroller features four I<sup>2</sup>Cs, six SPIs with three multiplexed simplex I<sup>2</sup>S, SDMMC, four USARTs, four UARTs, two CANs, three 12-bit ADCs, two 12-bit DACs, two SAIs, 8- to 14-bit digital camera module interface, internal 320+16+4-Kbyte SRAM and 1-Mbyte Flash memory, USB HS OTG, USB FS OTG, Ethernet MAC, FMC interface, Quad-SPI interface, SWD debugging support. This discovery board offers everything required for users to get started quickly and develop applications easily.    </p>  .......    <p>      <strong>&nbsp; </strong>    </p>  </div> |

The whole block I did not post here because there is a lot of text. I'll probably attach the whole zipped folder with all the files together with the project at the bottom of the page. And in general you can place any information in this block, as they say, to your taste and color.

Block **sidebar** :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <div id="sidebar">    <ul>      <li>Пункт 1</li>      <li>Пункт 2</li>      <li>Пункт 3</li>      <li>Пункт 4</li>    </ul>  <p>&nbsp;</p>  </div> |

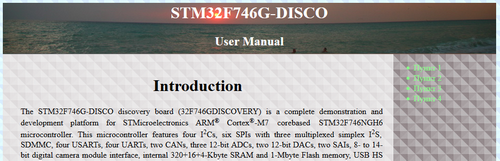
Here is just a list.

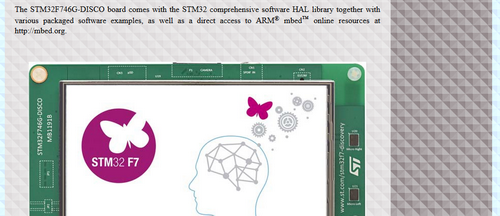
Well, block the **footer** :

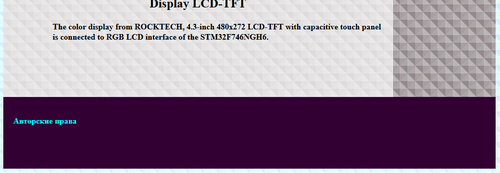


|  |  |
| --- | --- |
| 1  2  3  4  5 | <div id="footer">    <p>     <strong>Авторские права</strong>    </p>  </div> |

Save the changes in the page and open it in the browser







Our page has got a normal look.

It remains only to give functional.

Let's check the operation of our server for now.

Generate the file **fsdata.c** , update the project tree, disable the compilation of this file and try to assemble the project.

The project will now have to gather.

In the function of the query processing task, we add the request for the css-script file

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | else if (strncmp((char const \*)buf,"GET /style.css",14)==0)  {    fs\_open(&file, "/style.css");    netconn\_write(newconn, (const unsigned char\*)(file.data), (size\_t)file.len, NETCONN\_NOCOPY);    fs\_close(&file);  }  else if (strncmp((char const \*)buf,"GET /IMG/img01.jpg",18)==0) |

Also add requests for images of background images

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | else if (strncmp((char const \*)buf,"GET /IMG/bg01.png",17)==0)  {    fs\_open(&file, "/IMG/bg01.png");    netconn\_write(newconn, (const unsigned char\*)(file.data), (size\_t)file.len, NETCONN\_NOCOPY);    fs\_close(&file);  }  else if (strncmp((char const \*)buf,"GET /IMG/bg02.jpg",17)==0)  {    fs\_open(&file, "/IMG/bg02.jpg");    netconn\_write(newconn, (const unsigned char\*)(file.data), (size\_t)file.len, NETCONN\_NOCOPY);    fs\_close(&file);  }  else if (strncmp((char const \*)buf,"GET /IMG/bg03.png",17)==0)  {    fs\_open(&file, "/IMG/bg03.png");    netconn\_write(newconn, (const unsigned char\*)(file.data), (size\_t)file.len, NETCONN\_NOCOPY);    fs\_close(&file);  }  else |

We'll collect the code once again, we'll edit the controller and try to see our page in the browser by typing the IP-address of our server in the address bar.

We should get the same picture as when we open our page locally.

These were all just preparatory measures.

And the functionality for sending and receiving data between the client and the server, we will continue to write in the [**next part of**](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-2/) our lesson.

# STM Lesson 128. LAN8STM Lesson 128. LAN8742A. LWIP. NETCONN. HTTP. AJAX. Part 3

Posted on [August 24, 2018](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-3/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [FreeRTOS](http://narodstream.ru/freertos/) , [LAN](http://narodstream.ru/lan/) , [Programming AVR](http://narodstream.ru/rub_avr/)- [3 comments ↓](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-3/#comments)

[10 proven drawing techniquesFree to participate in a series of webinars and get your favorite painting lessons!To learn morenewdirections.ru](https://an.yandex.ru/count/BVu-OsQDJc850DC1CKt2ari00000ECgs7402I09Wl0Xe173mgPjnW07Cl-VExEgVyII80OF0sv4ra06cjhIyBvW1_AUrjowW0UwsjBmlg07cfhMtBhW1qE_sxXd00GBO0UQClmtW0UgOlWte0Go80WEW0fBxjnMv0lakJAiisZy6y0BKiP-Y2lW2We20W82W4O03xQhkeGw80zpicVi2c0FfZWAe0nAm0mIu1Fy1w0IW2lW4aBe1Y0MGkW6G1P-r0g05wpUe1Qali0MfBxW5pqx01V2P1yW5iEe1q0Mb5U05Ml050PW6sl2yDg06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKG078_l5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0Z9_TiAW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n2nar2m-540009VbrzRTGK0m0k0emN82u3Kam7P2_XH00SZ-yL8w0kGkW7m2mRe30AO3VYSPV8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0xKv1sm3W6X3m0000000F0_g0_ue-7CgEJ-uaW0?stat-id=3&test-tag=75866331701249&format-type=24&banner-test-tags=eyI2MzcxNTc3MjM1IjoiNzU4NjYzMDIzNDkzMTIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Earn remotely, working 2-3 hours.**Free** course "Specialist in Internet Advertising" for 10 days! Sign up before 10.09.18To learn moreprofiinternet.artur-grant.ru](https://an.yandex.ru/count/96I794O-mh850Dq1CKx2ari00000ECgs7402I09Wl0Xe173qmTYm0e01clpa0eW1YSEGWp6G0PAFlOyjc06sygd2Ag01b8-zZose0UZkgS8gk06qaQRQ6C010jW1YDIE4k01kgkO1kW19FW1lhtUlW680WIW0gpkd16v0lakJAiisZy6y0APhjIL0VW2We20W82W4O03xQd0K8W3kgYVq0UO0uaXi0C4k0J_0UW4ymBu1Dpz18W5tFq4a0NiWWQW1Qc_0QW5aQ81i0MHeW6u1Vt30S05zO0Oo0MWZGRG1Vf8u0K-y0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKIYHMFl5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YMjyu8W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n2Z4hQo-54009l1f5zRTGK0m0k0emN82u3Kam7P2_XHA95O-yL8w0lS_GJm2mQ83Bwzthu1w0m2c0t9icVo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AFXX__g_k98?stat-id=4&test-tag=75866331701249&format-type=24&banner-test-tags=eyI2MDU2MDI3MDgxIjoiNzU4NjYzMDIzNDkzMTIifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of the**](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-2/) lesson we continued to write the functionality of our client-server for sending and receiving data between the server and the client.

In this part of the lesson, we will continue the work begun and first try our text data sent to the client, somehow sort.

The standard output function does not know how to do this, so we add a function to sort above our **DynWebPageStr** function

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //---------------------------------------------------------------  void StrSort (char\* str)  {    typedef struct strarray{      char data[20][50];    } strarray\_ptr;  }  //--------------------------------------------------------------- |

In this function, we created a structure with one field - a two-dimensional character array. When such an array is wrapped in a structure, it will be easier to attach to it.

Use for sorting we will bubble method. I think many people know this method. We compare two adjacent rows or values, and one that is smaller (if we, of course, sort our data in ascending order), or it remains at the top, or goes up. So we come to the bottom and we have the fact that the biggest value we have is already below. Then we repeat this process, but we do not reach the bottom line, but we reach only the previous one. And so long as we have the smallest value will not be displaced upward, like an air bubble.

Let's create one more pointer to the line, which we got as an argument

C



|  |  |
| --- | --- |
| 1  2 | } strarray\_ptr;  char \*str\_tmp = (void\*) str; |

Let's add some more local variables

C



|  |  |
| --- | --- |
| 1  2  3  4 | char \*str\_tmp = (void\*) str;  char \*istr;  int cnt = 0, cnt\_pass=0, res = 0, i, j;  uint32\_t itemsize; |

Create a pointer to the data type of our structure and bind it to the global character buffer

C



|  |  |
| --- | --- |
| 1  2 | uint32\_t itemsize;  strarray\_ptr \*strarray1 = (void\*) str\_buf; |

Next, add a loop that, by the carriage return characters and the new line, breaks up our incoming string into rows and adds them to our array

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | strarray\_ptr \*strarray1 = (void\*) str\_buf;  //copy to array  while(1)  {  istr = strstr(str\_tmp,"rn");  itemsize = istr - str\_tmp;  if((itemsize>10)&&(itemsize<100))  {  strncpy(strarray1->data[cnt],str\_tmp,itemsize+2);  strarray1->data[cnt][itemsize+2] = 0;  str\_tmp+=itemsize+2;  }  else  {  break;  }  cnt++;  } |

Next, add a sorting cycle of the rows by the algorithm, which I explained above

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | cnt++;  }  //sort  cnt\_pass=cnt;  for(j=0;j<(cnt-1);j++)  {  for(i=0;i<(cnt\_pass-1);i++)  {  res = strcmp (strarray1->data[i], strarray1->data[i+1]);  if (res>0)  {  //swap  strcpy (strarray1->data[cnt],strarray1->data[i+1]);  strcpy (strarray1->data[i+1],strarray1->data[i]);  strcpy (strarray1->data[i],strarray1->data[cnt]);  }  }  cnt\_pass--;  } |

And then, on the contrary, we copy our lines from the array back to our line, only now they will be sorted in alphabetical order

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | cnt\_pass--;    }    //copy from array    str[0] = 0;    for(i=0;i<cnt;i++)    {      if (i==0) strcpy (str,strarray1->data[i]);      else strcat (str,strarray1->data[i]);    }    str\_tmp[0] = 0;  } |

At the end of the function body, we zero the end of the line.

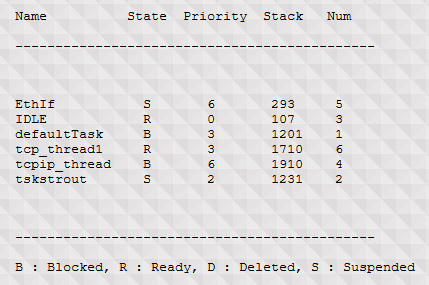
Now we call our string sort function in **DynWebPageStr**

C



|  |  |
| --- | --- |
| 1  2  3 | osThreadList((uint8\_t \*)(PAGE\_BODY + len));  StrSort((char \*)(PAGE\_BODY + len));  netconn\_write(conn, PAGE\_BODY, strlen((char\*)PAGE\_BODY), NETCONN\_COPY); |

We will collect the code, we will sew the controller, update the page in the browser and press the START button



Now we have sorted everything, and processes, whose name is from the big letter, are on top. Even somehow it looks static after we jumped our lines. But if there is a gradual increase in the use of the stack in one of the functions, then we will see this without difficulty.

While we have still transmitted text data from the server to the browser.

But it is also possible to transmit data of other types.

Often there is a need to transfer arrays of numeric and binary data that are stored in memory, not in the form of string expressions of these quantities, namely in a pristine binary form. It is not worthwhile to convert this data into a string first, and then on the client side, to convert them to numeric ones. Such actions take a lot of resources - both CPU time and memory. If there is a possibility to do without encoding-decoding, then it is better to use it. We always do this with you when we send readings from various sensors, as well as the state of any processes. Let's do this in our case.

Imagine the situation that we have an array of some data, for example, the ADC readings for a certain period of time. And we want to show this array on the page in the client's browser as a graph.

We, of course, will not connect and investigate any signals today, we will simply create an array of random variables, give them to the client and display them in a graph.

To add graphs to WEB-pages, there are several ready-made free libraries.

All of them use JavaScript and find them uncomplicated.

So we will use one of them - this is the **Chart.js** library , which is released under the **MIT license**. This license allows you to use this library for free for any purpose with only one remark - a link to the license file. We can also place the license file in the folder with your project, if we distribute such a project somewhere. We do not need to do this at all, since we download the library from GitHub as a JS script file, and the contents of this file have a link to the license file for Github. So, I think, we do not violate anything. So go to GitHub for the library by clicking https://github.com/chartjs/Chart.js and quietly download the script. It is distributed in several versions, we will take it minimized, since our virtual system does not like large files. In minimized versions of scripts, all tabs, spaces, and even line breaks are removed. From this, the file becomes at times half as large, although it becomes absolutely unreadable, but it does not suffer from this. The latest version of the library we will take on this link:[**Chart.js on GitHub**](https://github.com/chartjs/Chart.js/releases/latest) .

Go to the link and download the minimized version



Create in the folder with our main page the folder " **js** " and put this file there.

We connect it in our **index.php** main page , and at the same time add the title of our page

XHTML



|  |  |
| --- | --- |
| 1  2  3 | <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>STM32F746G-DISCO</title>  <script src="js/Chart.min.js"></script> |

Now we can safely access all the functions of this library. I will not tell you about them, since this is not a lesson, we will just use them. On the official website, all this is told.

In the **article section,** after our information about the processes that we just looked at, we will also add a **canvas** block , as the library requires, as well as a block with a START button similar to the button in the previous block that will launch our graph, for which we will add this button also a function

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <p></p>  <p></p>  <canvas id="myChart" width="700" height="400"></canvas>  <p></p>  <div id="butchart">  &nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startchart()" value="START"/>  </div>  <p></p> |

Before adding this function, we first add a class object to our script blocks

JavaScript



|  |  |
| --- | --- |
| 1  2 | var xhr;  var myChart; |

Then in the function **onload ()** we get a reference to the block with the identifier of our graph and create from 512 16-bit numbers

JavaScript



|  |  |
| --- | --- |
| 1  2  3 | xhr = new(XMLHttpRequest);  var ctx = document.getElementById("myChart");  var data16Array = new Uint16Array(512); |

Then we will create an object of the graph class, in which we will perform initial initialization. Since we do not need labels for each point, we will fill them with blank lines using the appropriate array, then adjust the background, add data in the form of our array with zeros, make the settings for colors, backgrounds and some other options that you can also see on the official website of the library developer

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | var data16Array = new Uint16Array(512);  myChart = new Chart(ctx, {  type: 'line',  data: {  labels: ["", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", ""],  datasets: [{  label: '# of Votes',  data: data16Array,  backgroundColor: 'rgba(150, 150, 0, 0.0)', //полностью прозрачный фон  borderColor: 'rgba(150, 150, 0,1)',  borderWidth: 2,  pointStyle: 'line',  pointRadius: 0  }]  },  options: {  tooltips: {  enabled: false  },  animation: {  duration: 0  },  responsive: true,  scales: {  xAxes: [{  ticks:{  min: 0,  max: 512,  stepSize : 64,  },  stacked: true,  gridLines: {  lineWidth: 0,  color: "rgba(255,255,255,0.0)"  }  }],  yAxes: [{  stacked: true,  ticks: {  min: 0,  max: 4096,  stepSize: 256,  }  }]  }  }  });  } |

Add one more timer, create an ID for it first, adding one more variable

JavaScript



|  |  |
| --- | --- |
| 1 | var idTimer1, idTimer2; |

After the function of the first **Timer1** timer **,** add the function for the second timer

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | idTimer1 = setTimeout("Timer1()", 1000);  }  function Timer2(){  xhr.open("GET", "content.bin?r=" + Math.random(), true); //Math.random() - защита от кеширования  xhr.responseType = "arraybuffer";  } |

In this function, we also created a query, but with a different type. This type is what we need to operate with raw (binary) data.

Then we add the event to the processing of the incoming data from the server, in which we consider our data as 16-bit and assign it to the data field of our graph, and then we update it. Then we send a request to the server and run our timer

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | xhr.responseType = "arraybuffer";  xhr.onload = function (oEvent) {  var uint16Array = new Uint16Array(this.response);  myChart.data.datasets[0].data = uint16Array;  myChart.update();  }  xhr.send(null);  idTimer2 = setTimeout("Timer2()", 1000);  } |

Add a handler function for our button at the bottom of the script

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function startchart(){  document.getElementById('butstring').innerHTML = '';  document.getElementById('butchart').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #666666;" onclick="stopchart()" value="STOP"/>';  Timer2();  }  </script> |

We reset the field with information about the processes, then we turned the START button into the STOP button, adding the corresponding function. And, of course, do not forget to run our timer, or rather its function. Let's write the functions for the STOP button below

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function stopchart(){  document.getElementById('butstring').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startstring()" value="START"/>';  document.getElementById('butchart').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startchart()" value="START"/>';  clearTimeout(idTimer2);  }  </script> |

Here we return the START button not only to our field, but also to the field for requesting information about the processes. Thus, during the timer operation, we block the restart of a given task, as well as another task.

Now we need to add similar locks to our previous functions for the output buttons from the process state server.

The **startstring ()** function

JavaScript



|  |  |
| --- | --- |
| 1  2 | function startstring(){  document.getElementById('butchart').innerHTML = ''; |

The **stopstring ()** function

JavaScript



|  |  |
| --- | --- |
| 1  2 | document.getElementById('butstring').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startstring()" value="START"/>';  document.getElementById('butchart').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startchart()" value="START"/>'; |

Now we save our page, generate the file **fsdata.c** , update the tree of our project and add one more function of generating the document to the client's request after the function **DynWebPageStr**

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | //---------------------------------------------------------------  void DynWebPage(struct netconn \*conn, uint16\_t y\_pos)  {    portCHAR PAGE\_BODY[1300];    uint16\_t len = 0;    int i;    PAGE\_BODY[0] = 0;    int val = 0;  }  //--------------------------------------------------------------- |

In this function, we have just added some local variables and an array for the future document, which we will send to the client on his request.

Collect the header for the HTTP packet

C



|  |  |
| --- | --- |
| 1  2  3 | int val = 0;  sprintf(PAGE\_BODY,"%s%s%s%s%drn%s",PAGE\_HEADER\_200\_OK,PAGE\_HEADER\_SERVER,PAGE\_HEADER\_CONTENT\_STREAM,        PAGE\_HEADER\_LEN,1024,PAGE\_HEADER\_BYTES); |

In this header, we passed the length in bytes to make it easier for the client to work with our package. We have only 512 values ​​of 16 bits, so the bytes will be 1024.

We will measure the length of the header, and in the cycle, we will type the data, forming them randomly, using for our hardware generator, and then transfer them together with the header to the client

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | sprintf(PAGE\_BODY,"%s%s%s%s%drn%s",PAGE\_HEADER\_200\_OK,PAGE\_HEADER\_SERVER,PAGE\_HEADER\_CONTENT\_STREAM,          PAGE\_HEADER\_LEN,1024,PAGE\_HEADER\_BYTES);    len = strlen(PAGE\_BODY);    for(i=0;i<512;i++)    {      val = HAL\_RNG\_GetRandomNumber(&hrng)%4096;      PAGE\_BODY[len + i \* 2] = (uint8\_t)val;      PAGE\_BODY[len + i \* 2 + 1] = (uint8\_t)(val>>8);    }    netconn\_write(conn, PAGE\_BODY, strlen((char\*)PAGE\_BODY) + 1024, NETCONN\_COPY);  } |

We will also add to the function of the query processing task to track the client's request, and in the condition body we will call our function of forming the package to respond to the client

C



|  |  |
| --- | --- |
| 1  2  3  4  5 | else if (strncmp((char const \*)buf,"GET /content.bin",16)==0)  {  DynWebPage(newconn, arg\_sock->y\_pos);  }  else |

Also, do not forget to add the client request to the library file for the graph

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | else if (strncmp((char const \*)buf,"GET /js/Chart.min.js",20)==0)  {  fs\_open(&file, "/js/Chart.min.js");  netconn\_write(newconn, (const unsigned char\*)(file.data), (size\_t)file.len, NETCONN\_NOCOPY);  fs\_close(&file);  }  else if (strncmp((char const \*)buf,"GET /IMG/img01.jpg",18)==0) |

We will collect the code, we will sew the controller, update the page in the browser and press the START button under the field for the graph



We have an excellent update once a second, I tried to shorten the timer interval to 40 milliseconds, that is, the schedule was updated to 25 frames per second, but sometimes there were suspicions, especially in a busy network. I did not try to connect the board directly to the computer directly. But, I believe that, with such speed, the transfer of 16-bit data and displaying such a large amount of data on the graph dynamically is a great thing.

I think that when we use WebSocket, the process will go even more fun.

So, in this lesson, we learned how to transfer data to the client on demand via the HTTP protocol, using some interesting features of the JavaScript language without reloading the page. And not only this, but also transfer data from the client to the server also without reloading the page. We also learned how to transfer packets via the HTTP protocol, not only string, but also binary types.

Thank you all for attention!

**742A. LWIP. NETCONN. HTTP. AJAX. Part 2**

Posted on [August 23, 2018](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-2/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [FreeRTOS](http://narodstream.ru/freertos/) , [LAN](http://narodstream.ru/lan/) , [Programming STM32](http://narodstream.ru/rub_stm32/)- [No Comments ↓](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-2/#respond)

[Earn remotely, working 2-3 hours.**Free** course "Specialist in Internet Advertising" for 10 days! Sign up before 10.09.18To learn moreprofiinternet.artur-grant.ru](https://an.yandex.ru/count/LQBGeKTe-H050E01CJJ2ari00000ECgs7402I09Wl0Xe173qmTYm0e01clpa0eW1YSEGWp6G0PAFlOyjc06sygd2Ag01b8-zZose0UZkgS8gk06qaQRQ6C010jW1YDIE4k01kgkO1kW19FW1lhtUlW680WEW0gpkd16v0k5ncgd32VONy0BgcUsH2VW2We20W82m2u03xQd0K8W3kgYVq0UO0uaXg0CIi0C4k0J_0UW4ZmBu1EcT18W5wPq4a0Nre0QW1SQY0QW5u8a1i0NWYG6u1S_B0S05zO0Oo0MWZGRG1Vuzu0K-y0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W000C1M0000gGVuKT1D3FN5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YMjyu8W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n3hjC9C-5400BQm4LvRTGK0m0k0emN82u3Kam7P2_XHq4qCzSL8w0lfdGJm2mQ83Bwzthu1w0m2c0tud6No3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AFHxRkX_U98?stat-id=3&test-tag=50027808448513&format-type=24&banner-test-tags=eyI2MDU2MDI3MDgxIjoiNTAwMjc3NzkwOTY1NzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Unclear errors of parents,for which the child pays. Free test for parentsTo learn moreyurkovskaya.com](https://an.yandex.ru/count/3TASVTswZMi50Da1CJJ2ari00000ECgs7402I09Wl0Xe172Civx21u01cfZa1OW1cFwAt3IG0RIeoAGlc07mnlx_BQ01jgZ8f2-e0PJ6_lyjk06m-jNW6S010jW1_AsU2-01vehT2kW1TlW10OW21A02sA_P5Ra2uN6QgSC9zXV80gEaokYRwD2EPV02fCY8rmpu0eA0W820i0k00_YDr4U80zEfbQS7i0C4k0J_0UW4iGBu19XSY0MON905WRW2e0NK6wW5k1Um1RWNk0MfJy05bSq9o0N0k0BG1SOAu0K9y0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00IS1M0000gGVuKHJ43VN5IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0Zir-mAW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n0PN-_e-54008qE4bvRTGK0m0k0emN82u3Kam7P2_XH5CGDzSL8w0kONF0B1eWChgZUlW7e30AO3ScoP_8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0x0X3sm3W6X3m0000000F0_g0_ueu7boQ7zuaW0?stat-id=4&test-tag=50027808448513&format-type=24&banner-test-tags=eyI2MzQ2NTc0MzYyIjoiNTAwMjc3NzkwOTY1NzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)18+

In the [**previous part of**](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-1/) our lesson we got acquainted with AJAX technology, customized the project, and also created a beautiful document to display in the browser, but so far without functionality.

Now add some functionality to our client-server.

Let's first learn how to transfer data from the browser to the server.

To do this, in the **sidebar** block **,** add a few multicolored buttons

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | </ul>    <p>&nbsp;</p>    <div class = "butcenter">      <input class = "butcolor" type="button" style="color: #00ffff; background-color: #ff0000;" onclick="red()" value="RED"/>      <br><br>      <input class = "butcolor" type="button" style="color: #ff00ff; background-color: #00ff00;" onclick="green()" value="GREEN"/>      <br><br>      <input class = "butcolor" type="button" style="color: #ffff00; background-color: #0000ff;" onclick="blue()" value="BLUE"/>      <br><br>      <input class = "butcolor" type="button" style="color: #ffffff; background-color: #000000;" onclick="black()" value="BLACK"/>      <br><br>      <input class = "butcolor" type="button" style="color: #ff0000; background-color: #00ffff;" onclick="cyan()" value="CYAN"/>      <br><br>      <input class = "butcolor" type="button" style="color: #00ff00; background-color: #ff00ff;" onclick="magenta()" value="MAGENTA"/>      <br><br>      <input class = "butcolor" type="button" style="color: #0000ff; background-color: #ffff00;" onclick="yellow()" value="YELLOW"/>      <br><br>      <input class = "butcolor" type="button" style="color: #000000; background-color: #ffffff;" onclick="white()" value="WHITE"/>    </div>  </div> |

We assigned a class to a block with buttons, also to the buttons themselves, also defined their color, the color of their text, set the text itself, and also assigned function names to handle the events of pressing the buttons.

Save the page and see how it looks before styling.



I think that this is not serious.

Let's add some styles to the block with buttons and buttons in the **style.css** file

CSS



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | .butcenter {    text-align: center;  }    .butcolor {    width: 150px;    height: 48px;    font-size: 17px;  } |

Save the file with the styles and update the page



Now that's another matter. We have not yet strongly stylized the buttons, we could still be puffed up, but the purpose of the lesson is not this. At least on such buttons it will be much more pleasant to press.

The purpose of these buttons - by clicking on them will appear on the display board square with the same color as the button.

Well now, finally the long awaited AJAX.

So that we do not have to take our script to the very bottom of the page, in the body tag of the document we add here a handler with the name of the function that will be called after the entire document is loaded

XHTML



|  |  |
| --- | --- |
| 1 | <body onload="onload()"> |

This is necessary in order to run our scripts after the entire document is loaded so that the document load does not "slow down".

We will not put our script into a separate file, although we could, but we must first learn how to work in a document file.

Therefore, also in the tags **head** create this block for the script

XHTML



|  |  |
| --- | --- |
| 1  2  3 | <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <script>  </script> |

Now between the opening and closing tag, we will write the text of our JavaScript.

Let's start with the handler function, which will be launched after the document is loaded.

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <script>    var xhr;    function onload(){      xhr = new(XMLHttpRequest);    }  </script> |

While we just created an instance of the HTTP request object to the server. Just such an object allows you to make requests without reloading the entire page.

Before the function, we created a global variable for the query, so that we could then access it in other functions.

Therefore, in our function, the page load handler will only create a query object.

Accordingly, we will not go into the details of processing these requests, we do not have a JavaScript lesson.

But we will create a query by clicking on one of the buttons, and then also in other functions.

Let's start with red

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | xhr = new(XMLHttpRequest);  }  function red(){    xhr.open("GET", "color.html?c=1", true);    xhr.responseType = "text";    xhr.send(null);  } |

We will send a GET request to the server with the appropriate string, then assign the query type "text", and then directly call the function to send our request to the server.

Similar functions will be added for other buttons

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | function green(){      xhr.open("GET", "color.html?c=2", true);      xhr.responseType = "text";      xhr.send(null);    }    function blue(){      xhr.open("GET", "color.html?c=3", true);      xhr.responseType = "text";      xhr.send(null);    }    function black(){      xhr.open("GET", "color.html?c=4", true);      xhr.responseType = "text";      xhr.send(null);    }    function cyan(){      xhr.open("GET", "color.html?c=5", true);      xhr.responseType = "text";      xhr.send(null);    }    function magenta(){      xhr.open("GET", "color.html?c=6", true);      xhr.responseType = "text";      xhr.send(null);    }    function yellow(){      xhr.open("GET", "color.html?c=7", true);      xhr.responseType = "text";      xhr.send(null);    }    function white(){      xhr.open("GET", "color.html?c=8", true);      xhr.responseType = "text";      xhr.send(null);    }  </script> |

Generate the file **fsdata.c** and update the tree of our project, only before collecting the code, we will need to process our requests.

Again, in the function of the query processing task, add the data requests. You can do this with the help of the variant operator, since in our requests only the last character will be different

C



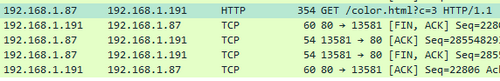
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | else if (strncmp((char const \*)buf,"GET /color.html?c=",18)==0)  {    switch (buf[18])    {      case '1':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_RED);        break;      case '2':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_GREEN);        break;      case '3':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_BLUE);        break;      case '4':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_BLACK);        break;      case '5':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_CYAN);        break;      case '6':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_MAGENTA);        break;      case '7':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_YELLOW);        break;      case '8':        TFT\_FillRectangle(100,120,200,220,LCD\_COLOR\_WHITE);        break;    }  }  else |

We will collect the code, we will impose the controller, update our page in the browser, using the same IP, and press on our buttons. We should see as reaction the appearing square of a certain color on the display of the debug board





Let's also look at WireShark, that no page is reloaded in response to a client request



Yes, we know this without analysis, we did not give anything to the server in response to his request.

The next task is to take something from the server. For this, we must also make a request. Without a request, the server to the client can transfer data only when using WebSocket technology, but we will soon reach it.

Add in the code of our page index.html in any place (for example, between the first two pictures here is such a code

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <img src="IMG/img01.jpg">  </p>  <p></p>  <p></p>  <pre><br>Name State Priority Stack Num    <br>---------------------------------------------<br>  </pre>  <pre id="information"></pre>  <pre><br><br>---------------------------------------------    <br>B : Blocked, R : Ready, D : Deleted, S : Suspended<br>  </pre>  <p></p> |

I think you know that the <pre> tag is used to accurately convey text, so that the spaces are also spaces, not ignored, etc.

An empty **pre** block with the **information** identifier will serve to ensure that the data from the server arrives here.

Also after all this, let's add one more button for starting requests for information from the server, it will also serve to stop the transmission

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <p></p>  <div id="butstring">    &nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startstring()" value="START"/>  </div>  <p></p>  <p></p> |

We put the button in the block to make it more convenient to go there.

Now let's move on to our script and add another variable, we'll see later what it's used for

JavaScript



|  |  |
| --- | --- |
| 1  2 | var xhr;  var idTimer1; |

Add a function for the query.

Since requests will be periodic, we will give her the appropriate name

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | function Timer1(){    xhr.open("GET", "content.html?r=" + Math.random(), true); //Math.random() - защита от кеширования    xhr.responseType = "text";    xhr.onload = function (oEvent) {      document.getElementById('information').innerHTML = xhr.response;    }    xhr.send(null);    idTimer1 = setTimeout("Timer1()", 1000);  }  function red(){ |

We sent a similar request to the server, but since we will receive data from it, that is the danger that they cached and the browser starts to cache information from the cache, so we'll add a random number to the query so that the browser does not seem to come alone and the same data. Next, we also assign a data type to "text," and then add a handler function, in JavaScript, we can write the body of this handler right here. In the body, we send the received text information from the server to our block, then we call the object method to send the request to the server, and when finished we start the one-time timer, in which we add the name of the function as the first argument, which we will call after the interval in milliseconds, added in the second argument. But, since we have the name of the function in which we are, then the timer we will get is not a one-off, but quite a cyclical one. Also we will use the returned argument of the function and assign our timer an identifier so that we can stop it at any time.

Only in itself the first time this function will not be called, we will call it in the handler of our button, which we just added. Add this handler at the bottom of the script

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5 | function startstring(){  document.getElementById('butstring').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #666666;" onclick="stopstring()" value="STOP"/>';  Timer1();  }  </script> |

Before starting the timer by clicking the button, we first change the block with the button, in which we turn this button into the stop button of the process of requests to the server, and also assign it another handler. We'll then run our timer.

So let's now also add a handler for the stop button

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function stopstring(){    document.getElementById('butstring').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startstring()" value="START"/>';    clearTimeout(idTimer1);    document.getElementById('information').innerHTML = "";  }  function startstring(){ |

Here we again turn the button into the start button, stop the timer and clear the window with the information.

We save our page, generate the file **fsdata.c** , update the tree of our project, and after the function of the task of displaying the lines on the display, **TaskStringOut** we add a function for generating a dynamic web document for replying to the client on the request

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //---------------------------------------------------------------    void DynWebPageStr(struct netconn \*conn)  {    portCHAR PAGE\_BODY[768];    uint16\_t len = 0;  }  //--------------------------------------------------------------- |

In this function, we only pass a pointer to the structure of the connection.

Now we trace the client's request to the function of the query processing task, in the body of the condition we will call our function to generate a response

C



|  |  |
| --- | --- |
| 1  2  3  4  5 | else if (strncmp((char const \*)buf,"GET /content.html",17)==0)  {    DynWebPageStr(newconn);  }  else |

Before writing the body of the function of forming the response to the client, we will add several global arrays with the fragments of the HTTP packet header. We will place them in the FLASH-memory of the controller using special directives

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | #define MAIL\_SIZE (uint32\_t) 5  static const unsigned char PAGE\_HEADER\_200\_OK[] = {    //"HTTP/1.1 200 OK"    0x48,0x54,0x54,0x50,0x2f,0x31,0x2e,0x30,0x20,0x32,0x30,0x30,0x20,0x4f,0x4b,0x0d,    0x0a,    //zero    0x00  };  static const unsigned char PAGE\_HEADER\_SERVER[] = {    //"Server: lwIP/1.3.1 (http://savannah.nongnu.org/projects/lwip)"    0x53,0x65,0x72,0x76,0x65,0x72,0x3a,0x20,0x6c,0x77,0x49,0x50,0x2f,0x31,0x2e,0x33,    0x2e,0x31,0x20,0x28,0x68,0x74,0x74,0x70,0x3a,0x2f,0x2f,0x73,0x61,0x76,0x61,0x6e,    0x6e,0x61,0x68,0x2e,0x6e,0x6f,0x6e,0x67,0x6e,0x75,0x2e,0x6f,0x72,0x67,0x2f,0x70,    0x72,0x6f,0x6a,0x65,0x63,0x74,0x73,0x2f,0x6c,0x77,0x69,0x70,0x29,0x0d,0x0a,    //zero    0x00  };  static const unsigned char PAGE\_HEADER\_CONTENT\_TEXT[] = {    //"Content-type: text/html"    0x43,0x6f,0x6e,0x74,0x65,0x6e,0x74,0x2d,0x74,0x79,0x70,0x65,0x3a,0x20,0x74,0x65,    0x78,0x74,0x2f,0x68,0x74,0x6d,0x6c,0x0d,0x0a,0x0d,0x0a,    //zero    0x00  };  //\*  static const unsigned char PAGE\_HEADER\_CONTENT\_STREAM[] = {    //"Content-Type: application/octet-stream"    0x43,0x6f,0x6e,0x74,0x65,0x6e,0x74,0x2d,0x54,0x79,0x70,0x65,0x3a,0x20,0x61,0x70,    0x70,0x6c,0x69,0x63,0x61,0x74,0x69,0x6f,0x6e,0x2f,0x6f,0x63,0x74,0x65,0x74,0x2d,    0x73,0x74,0x72,0x65,0x61,0x6d,0x0d,0x0a,    //zero    0x00  };  static const unsigned char PAGE\_HEADER\_LEN[] = {    //"Content-Length: "    0x43,0x6f,0x6e,0x74,0x65,0x6e,0x74,0x2d,0x4c,0x65,0x6e,0x67,0x74,0x68,0x3a,0x20,    //zero    0x00  };  static const unsigned char PAGE\_HEADER\_BYTES[] = {    //"Accept-Ranges: bytes"    0x41,0x63,0x63,0x65,0x70,0x74,0x2d,0x52,0x61,0x6e,0x67,0x65,0x73,0x3a,0x20,0x62,    0x79,0x74,0x65,0x73,0x0d,0x0a,0x0d,0x0a,    //zero    0x00  };  /\* USER CODE END PV \*/ |

In these arrays, we have exactly the character codes.

Now we continue to write the body of the function **DynWebPageStr** .

We start to prepare in it a line for sending to the client

C



|  |  |
| --- | --- |
| 1  2 | uint16\_t len = 0;  sprintf(PAGE\_BODY,"%s%s%s",PAGE\_HEADER\_200\_OK,PAGE\_HEADER\_SERVER,PAGE\_HEADER\_CONTENT\_TEXT); |

We have prepared for the time being only a headline.

Send to the client we will information about the processes. And, since requests will occur regularly, this information on the page will be updated dynamically.

Request information about the processes and bring it right to the right place

C



|  |  |
| --- | --- |
| 1  2  3 | sprintf(PAGE\_BODY,"%s%s%s",PAGE\_HEADER\_200\_OK,PAGE\_HEADER\_SERVER,PAGE\_HEADER\_CONTENT\_TEXT);  len = strlen(PAGE\_BODY);  osThreadList((uint8\_t \*)(PAGE\_BODY + len)); |

Well and at last we will pass our document to the client

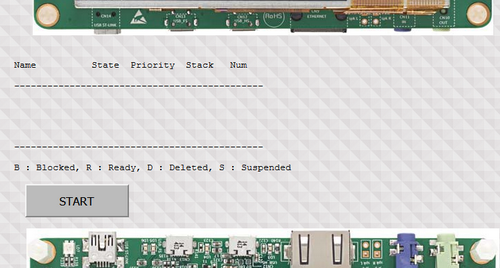
C



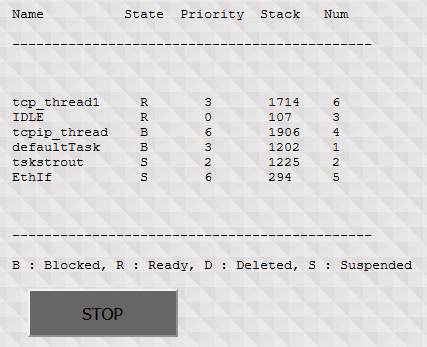
|  |  |
| --- | --- |
| 1  2  3 | osThreadList((uint8\_t \*)(PAGE\_BODY + len));    netconn\_write(conn, PAGE\_BODY, strlen((char\*)PAGE\_BODY), NETCONN\_COPY);  } |

We will collect the code, we will tell the controller and see the result in the browser.

When you start the page, we see such a picture

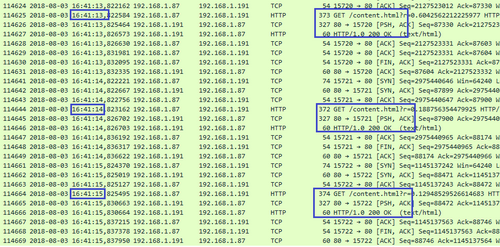


Press the START button and see that we will have the information and then it will be slowly updated (once a second). True use of the stack here is shown maximum, so the information will not change much



Although due to the fact that the information is not sorted in any way, the lines with the tasks will jump from place to place, creating the illusion that the information on the lines is changing. In fact, if we sort it, we'll notice that it changes very rarely. But, most importantly, what we see is that the information sent from the server to the client is updated regularly. We sort the lines a bit later.

Although we already perfectly see that the page we do not every time reloads, however, we will see this in WireShark (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2018/08/stm128img10.png)

Of course, if we really had to monitor dynamically changing data, then it would be very difficult for us to do this in such a regime, when the lines jump up and down, so we try to sort our data alphabetically.

Only we will do this in the [**next part of**](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-3/) our lesson, in which we will also work with the transfer of data of a binary type and display them in the browser as a graph.

**STM Lesson 128. LAN8742A. LWIP. NETCONN. HTTP. AJAX. Part 3**

Posted on [August 24, 2018](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-3/)by [http://1.gravatar.com/avatar/4824b24065500834db4b9f331b608833?s=32&d=mm&r=gNarod Stream](http://narodstream.ru/author/admin/) Posted in [FreeRTOS](http://narodstream.ru/freertos/) , [LAN](http://narodstream.ru/lan/) , [Programming AVR](http://narodstream.ru/rub_avr/)- [3 comments ↓](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-3/#comments)

[10 proven drawing techniquesFree to participate in a series of webinars and get your favorite painting lessons!To learn morenewdirections.ru](https://an.yandex.ru/count/NBjw-Ja-HAK50DC1CUZ2ari00000ECgs7402I09Wl0Xe173mgPjnW07Cl-VExEgVyII80OF0sv4ra06cjhIyBvW1_AUrjowW0UwsjBmlg07cfhMtBhW1qE_sxXd00GBO0UQClmtW0UgOlWte0Go80WEW0fBxjnMv0lakJAiisZy6y0BNyyUK2FW2We20W82G5803xQhkeGw80zpicVi2c0FfZWAe0nAm0mIu1Fy1w0IR2VW4fya1Y0MdoG6G1Usr0g05iZoe1PSpi0MNCxW5uqx01V2P1yW5iEe1q0M35-05Ml050PW6leccDg06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKNXkx1_6IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0Z9_TiAW0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mQg2n1MSwaw-54002AvoMXRTGK0m0k0emN82u3Kam7P2_XHU6xi7yP8w0kdoG7m2mRe30AO3VYSPV8D0FeD088E08aE00000000y3-G3i24FPWEnjVCr9M9uBeJe0xKv1sm3W6X3m0000000F0_g0_uey6LkVs7uqW0?stat-id=3&test-tag=88510715420673&format-type=24&banner-test-tags=eyI2MzcxNTc3MjM1IjoiODg1MTA2ODYwNjg3MzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

[Earn remotely, working 2-3 hours.**Free** course "Specialist in Internet Advertising" for 10 days! Sign up before 10.09.18To learn moreprofiinternet.artur-grant.ru](https://an.yandex.ru/count/2pomg00yP2a50Dq1CUZ2ari00000ECgs7402I09Wl0Xe173qmTYm0e01clpa0eW1YSEGWp6G0PAFlOyjc06sygd2Ag01b8-zZose0UZkgS8gk06qaQRQ6C010jW1YDIE4k01kgkO1kW19FW1lhtUlW680WIW0gpkd16v0lakJAiisZy6y0AQkuYb2_W2We20W82G5803xQd0K8W3kgYVq0UO0uaXi0C4k0J_0UW4wGBu1EgA18W5wee4a0M9YmQW1UsS0QW5zeG1i0NsX06u1OZ60S05zO0Oo0MWZGRG1UWxu0K-y0K1c0Q0qApp3g06xWAe1ku2oGR6AvDQIO1r4z46nh_8qKc0THFP1W00071n0000gGVuKLGL8o36IB07W82G3D070k07XWhu1m60207G2BgAW870a802u0YMjyu8W0e1mGe00000003mFzWA0k0AW8bw-0h0_1M82mYg2n1QATVc-5400F1YrsXRTGK0m0k0emN82u3Kam7P2_XHL1KZ8CP8w0lgYWJm2mQ83Bwzthu1w0m2c0t9icVo3G3w3G223W293W0000000F0_a0x0X3sO3iRNpDILYU2w4w0Em8Gzi0u1eGy00000003mFwWF-AE1aAc4YED8?stat-id=4&test-tag=88510715420673&format-type=24&banner-test-tags=eyI2MDU2MDI3MDgxIjoiODg1MTA2ODYwNjg3MzYifQ%3D%3D&)[Yandex.Direct](https://direct.yandex.ru/?partner)

In the [**previous part of the**](http://narodstream.ru/stm-urok-128-lan8742a-lwip-netconn-http-ajax-chast-2/) lesson we continued to write the functionality of our client-server for sending and receiving data between the server and the client.

In this part of the lesson, we will continue the work begun and first try our text data sent to the client, somehow sort.

The standard output function does not know how to do this, so we add a function to sort above our **DynWebPageStr** function

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //---------------------------------------------------------------  void StrSort (char\* str)  {    typedef struct strarray{      char data[20][50];    } strarray\_ptr;  }  //--------------------------------------------------------------- |

In this function, we created a structure with one field - a two-dimensional character array. When such an array is wrapped in a structure, it will be easier to attach to it.

Use for sorting we will bubble method. I think many people know this method. We compare two adjacent rows or values, and one that is smaller (if we, of course, sort our data in ascending order), or it remains at the top, or goes up. So we come to the bottom and we have the fact that the biggest value we have is already below. Then we repeat this process, but we do not reach the bottom line, but we reach only the previous one. And so long as we have the smallest value will not be displaced upward, like an air bubble.

Let's create one more pointer to the line, which we got as an argument

C



|  |  |
| --- | --- |
| 1  2 | } strarray\_ptr;  char \*str\_tmp = (void\*) str; |

Let's add some more local variables

C



|  |  |
| --- | --- |
| 1  2  3  4 | char \*str\_tmp = (void\*) str;  char \*istr;  int cnt = 0, cnt\_pass=0, res = 0, i, j;  uint32\_t itemsize; |

Create a pointer to the data type of our structure and bind it to the global character buffer

C



|  |  |
| --- | --- |
| 1  2 | uint32\_t itemsize;  strarray\_ptr \*strarray1 = (void\*) str\_buf; |

Next, add a loop that, by the carriage return characters and the new line, breaks up our incoming string into rows and adds them to our array

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | strarray\_ptr \*strarray1 = (void\*) str\_buf;  //copy to array  while(1)  {  istr = strstr(str\_tmp,"rn");  itemsize = istr - str\_tmp;  if((itemsize>10)&&(itemsize<100))  {  strncpy(strarray1->data[cnt],str\_tmp,itemsize+2);  strarray1->data[cnt][itemsize+2] = 0;  str\_tmp+=itemsize+2;  }  else  {  break;  }  cnt++;  } |

Next, add a sorting cycle of the rows by the algorithm, which I explained above

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | cnt++;  }  //sort  cnt\_pass=cnt;  for(j=0;j<(cnt-1);j++)  {  for(i=0;i<(cnt\_pass-1);i++)  {  res = strcmp (strarray1->data[i], strarray1->data[i+1]);  if (res>0)  {  //swap  strcpy (strarray1->data[cnt],strarray1->data[i+1]);  strcpy (strarray1->data[i+1],strarray1->data[i]);  strcpy (strarray1->data[i],strarray1->data[cnt]);  }  }  cnt\_pass--;  } |

And then, on the contrary, we copy our lines from the array back to our line, only now they will be sorted in alphabetical order

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | cnt\_pass--;    }    //copy from array    str[0] = 0;    for(i=0;i<cnt;i++)    {      if (i==0) strcpy (str,strarray1->data[i]);      else strcat (str,strarray1->data[i]);    }    str\_tmp[0] = 0;  } |

At the end of the function body, we zero the end of the line.

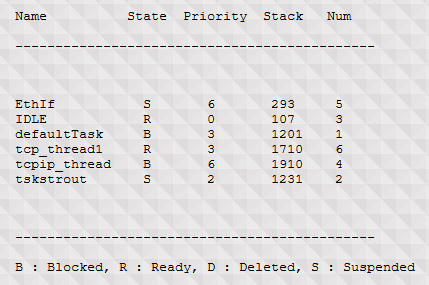
Now we call our string sort function in **DynWebPageStr**

C



|  |  |
| --- | --- |
| 1  2  3 | osThreadList((uint8\_t \*)(PAGE\_BODY + len));  StrSort((char \*)(PAGE\_BODY + len));  netconn\_write(conn, PAGE\_BODY, strlen((char\*)PAGE\_BODY), NETCONN\_COPY); |

We will collect the code, we will sew the controller, update the page in the browser and press the START button



Now we have sorted everything, and processes, whose name is from the big letter, are on top. Even somehow it looks static after we jumped our lines. But if there is a gradual increase in the use of the stack in one of the functions, then we will see this without difficulty.

While we have still transmitted text data from the server to the browser.

But it is also possible to transmit data of other types.

Often there is a need to transfer arrays of numeric and binary data that are stored in memory, not in the form of string expressions of these quantities, namely in a pristine binary form. It is not worthwhile to convert this data into a string first, and then on the client side, to convert them to numeric ones. Such actions take a lot of resources - both CPU time and memory. If there is a possibility to do without encoding-decoding, then it is better to use it. We always do this with you when we send readings from various sensors, as well as the state of any processes. Let's do this in our case.

Imagine the situation that we have an array of some data, for example, the ADC readings for a certain period of time. And we want to show this array on the page in the client's browser as a graph.

We, of course, will not connect and investigate any signals today, we will simply create an array of random variables, give them to the client and display them in a graph.

To add graphs to WEB-pages, there are several ready-made free libraries.

All of them use JavaScript and find them uncomplicated.

So we will use one of them - this is the **Chart.js** library , which is released under the **MIT license**. This license allows you to use this library for free for any purpose with only one remark - a link to the license file. We can also place the license file in the folder with your project, if we distribute such a project somewhere. We do not need to do this at all, since we download the library from GitHub as a JS script file, and the contents of this file have a link to the license file for Github. So, I think, we do not violate anything. So go to GitHub for the library by clicking https://github.com/chartjs/Chart.js and quietly download the script. It is distributed in several versions, we will take it minimized, since our virtual system does not like large files. In minimized versions of scripts, all tabs, spaces, and even line breaks are removed. From this, the file becomes at times half as large, although it becomes absolutely unreadable, but it does not suffer from this. The latest version of the library we will take on this link:[**Chart.js on GitHub**](https://github.com/chartjs/Chart.js/releases/latest) .

Go to the link and download the minimized version



Create in the folder with our main page the folder " **js** " and put this file there.

We connect it in our **index.php** main page , and at the same time add the title of our page

XHTML



|  |  |
| --- | --- |
| 1  2  3 | <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  <title>STM32F746G-DISCO</title>  <script src="js/Chart.min.js"></script> |

Now we can safely access all the functions of this library. I will not tell you about them, since this is not a lesson, we will just use them. On the official website, all this is told.

In the **article section,** after our information about the processes that we just looked at, we will also add a **canvas** block , as the library requires, as well as a block with a START button similar to the button in the previous block that will launch our graph, for which we will add this button also a function

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <p></p>  <p></p>  <canvas id="myChart" width="700" height="400"></canvas>  <p></p>  <div id="butchart">  &nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startchart()" value="START"/>  </div>  <p></p> |

Before adding this function, we first add a class object to our script blocks

JavaScript



|  |  |
| --- | --- |
| 1  2 | var xhr;  var myChart; |

Then in the function **onload ()** we get a reference to the block with the identifier of our graph and create from 512 16-bit numbers

JavaScript



|  |  |
| --- | --- |
| 1  2  3 | xhr = new(XMLHttpRequest);  var ctx = document.getElementById("myChart");  var data16Array = new Uint16Array(512); |

Then we will create an object of the graph class, in which we will perform initial initialization. Since we do not need labels for each point, we will fill them with blank lines using the appropriate array, then adjust the background, add data in the form of our array with zeros, make the settings for colors, backgrounds and some other options that you can also see on the official website of the library developer

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75 | var data16Array = new Uint16Array(512);  myChart = new Chart(ctx, {  type: 'line',  data: {  labels: ["", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",  "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", ""],  datasets: [{  label: '# of Votes',  data: data16Array,  backgroundColor: 'rgba(150, 150, 0, 0.0)', //полностью прозрачный фон  borderColor: 'rgba(150, 150, 0,1)',  borderWidth: 2,  pointStyle: 'line',  pointRadius: 0  }]  },  options: {  tooltips: {  enabled: false  },  animation: {  duration: 0  },  responsive: true,  scales: {  xAxes: [{  ticks:{  min: 0,  max: 512,  stepSize : 64,  },  stacked: true,  gridLines: {  lineWidth: 0,  color: "rgba(255,255,255,0.0)"  }  }],  yAxes: [{  stacked: true,  ticks: {  min: 0,  max: 4096,  stepSize: 256,  }  }]  }  }  });  } |

Add one more timer, create an ID for it first, adding one more variable

JavaScript



|  |  |
| --- | --- |
| 1 | var idTimer1, idTimer2; |

After the function of the first **Timer1** timer **,** add the function for the second timer

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | idTimer1 = setTimeout("Timer1()", 1000);  }  function Timer2(){  xhr.open("GET", "content.bin?r=" + Math.random(), true); //Math.random() - защита от кеширования  xhr.responseType = "arraybuffer";  } |

In this function, we also created a query, but with a different type. This type is what we need to operate with raw (binary) data.

Then we add the event to the processing of the incoming data from the server, in which we consider our data as 16-bit and assign it to the data field of our graph, and then we update it. Then we send a request to the server and run our timer

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | xhr.responseType = "arraybuffer";  xhr.onload = function (oEvent) {  var uint16Array = new Uint16Array(this.response);  myChart.data.datasets[0].data = uint16Array;  myChart.update();  }  xhr.send(null);  idTimer2 = setTimeout("Timer2()", 1000);  } |

Add a handler function for our button at the bottom of the script

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function startchart(){  document.getElementById('butstring').innerHTML = '';  document.getElementById('butchart').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #666666;" onclick="stopchart()" value="STOP"/>';  Timer2();  }  </script> |

We reset the field with information about the processes, then we turned the START button into the STOP button, adding the corresponding function. And, of course, do not forget to run our timer, or rather its function. Let's write the functions for the STOP button below

JavaScript



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | function stopchart(){  document.getElementById('butstring').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startstring()" value="START"/>';  document.getElementById('butchart').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startchart()" value="START"/>';  clearTimeout(idTimer2);  }  </script> |

Here we return the START button not only to our field, but also to the field for requesting information about the processes. Thus, during the timer operation, we block the restart of a given task, as well as another task.

Now we need to add similar locks to our previous functions for the output buttons from the process state server.

The **startstring ()** function

JavaScript



|  |  |
| --- | --- |
| 1  2 | function startstring(){  document.getElementById('butchart').innerHTML = ''; |

The **stopstring ()** function

JavaScript



|  |  |
| --- | --- |
| 1  2 | document.getElementById('butstring').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startstring()" value="START"/>';  document.getElementById('butchart').innerHTML = '&nbsp;&nbsp;&nbsp;&nbsp;<input class = "butcolor" type="button" style="color: #000000; background-color: #bbbbbb;" onclick="startchart()" value="START"/>'; |

Now we save our page, generate the file **fsdata.c** , update the tree of our project and add one more function of generating the document to the client's request after the function **DynWebPageStr**

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | //---------------------------------------------------------------  void DynWebPage(struct netconn \*conn, uint16\_t y\_pos)  {    portCHAR PAGE\_BODY[1300];    uint16\_t len = 0;    int i;    PAGE\_BODY[0] = 0;    int val = 0;  }  //--------------------------------------------------------------- |

In this function, we have just added some local variables and an array for the future document, which we will send to the client on his request.

Collect the header for the HTTP packet

C



|  |  |
| --- | --- |
| 1  2  3 | int val = 0;  sprintf(PAGE\_BODY,"%s%s%s%s%drn%s",PAGE\_HEADER\_200\_OK,PAGE\_HEADER\_SERVER,PAGE\_HEADER\_CONTENT\_STREAM,        PAGE\_HEADER\_LEN,1024,PAGE\_HEADER\_BYTES); |

In this header, we passed the length in bytes to make it easier for the client to work with our package. We have only 512 values ​​of 16 bits, so the bytes will be 1024.

We will measure the length of the header, and in the cycle, we will type the data, forming them randomly, using for our hardware generator, and then transfer them together with the header to the client

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | sprintf(PAGE\_BODY,"%s%s%s%s%drn%s",PAGE\_HEADER\_200\_OK,PAGE\_HEADER\_SERVER,PAGE\_HEADER\_CONTENT\_STREAM,          PAGE\_HEADER\_LEN,1024,PAGE\_HEADER\_BYTES);    len = strlen(PAGE\_BODY);    for(i=0;i<512;i++)    {      val = HAL\_RNG\_GetRandomNumber(&hrng)%4096;      PAGE\_BODY[len + i \* 2] = (uint8\_t)val;      PAGE\_BODY[len + i \* 2 + 1] = (uint8\_t)(val>>8);    }    netconn\_write(conn, PAGE\_BODY, strlen((char\*)PAGE\_BODY) + 1024, NETCONN\_COPY);  } |

We will also add to the function of the query processing task to track the client's request, and in the condition body we will call our function of forming the package to respond to the client

C



|  |  |
| --- | --- |
| 1  2  3  4  5 | else if (strncmp((char const \*)buf,"GET /content.bin",16)==0)  {  DynWebPage(newconn, arg\_sock->y\_pos);  }  else |

Also, do not forget to add the client request to the library file for the graph

C



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | else if (strncmp((char const \*)buf,"GET /js/Chart.min.js",20)==0)  {  fs\_open(&file, "/js/Chart.min.js");  netconn\_write(newconn, (const unsigned char\*)(file.data), (size\_t)file.len, NETCONN\_NOCOPY);  fs\_close(&file);  }  else if (strncmp((char const \*)buf,"GET /IMG/img01.jpg",18)==0) |

We will collect the code, we will sew the controller, update the page in the browser and press the START button under the field for the graph



We have an excellent update once a second, I tried to shorten the timer interval to 40 milliseconds, that is, the schedule was updated to 25 frames per second, but sometimes there were suspicions, especially in a busy network. I did not try to connect the board directly to the computer directly. But, I believe that, with such speed, the transfer of 16-bit data and displaying such a large amount of data on the graph dynamically is a great thing.

I think that when we use WebSocket, the process will go even more fun.

So, in this lesson, we learned how to transfer data to the client on demand via the HTTP protocol, using some interesting features of the JavaScript language without reloading the page. And not only this, but also transfer data from the client to the server also without reloading the page. We also learned how to transfer packets via the HTTP protocol, not only string, but also binary types.

Thank you all for attention!