

PROJECT 2

Making Web Services in C

Overview

In this lesson, we will develop RESTful APIs in C. This might be interesting as usually, web services or RESTful APIs (as they are so called) are usually made using an Object-Oriented Programming language like Ruby, Java or Python. Any programming language can be used to do any task when needed but the question we need to ask is: Does it make our work easier? After all, that is the thing which program or coding intends to do, make our life easier.

Even though C is not primarily made for developing Web Services which can respond to calls over the network for HTTP methods like GET, POST, PUT or DELETE; we can still do so using some external libraries for demonstration purposes. We say demonstration purposes because it is very less probable that you will actually do so in real-life project but just so that you know, it is possible (just like everything else) and we will do it.

What will we use?

1. **Ubuntu:** To develop Web Services in C, we will use the Ubuntu platform but please note, you can make use of any other platform equally well, like Windows or a Macintosh. The commands shown in this lesson will be for Ubuntu platform but these can have some equal counterparts for other mentioned platforms as well.
2. **Onion:** We will use an external third party library called Onion. The library is dual licensed under the Apache2 license and GPLv2+, so we can make almost anything with it, use it in our commercial and free software programs, and modify it to our needs. This is a simple library which can be used to make Web Services in C with easy callback mechanism (more on this in next section).

What is Callback Mechanism?

Before we start making Web Services in C and look at how Onion library actually works, we need to understand what is callback mechanism and how does it work.

A callback is any executable code that is passed as an argument to other code, which is expected to call back (execute) the argument at a given time [Source : Wiki]. In simple language, If the reference of a function is passed to another function as an argument to call it, then it it will be called as a Callback function. Let us consider an example here:

```
void populate_array(int *array, size_t arraySize, int (*getNextValue)(void))
{
    for (size_t i=0; i<arraySize; i++)
```

```

        array[i] = getNextValue();
    }

    int getNextRandomValue(void)
    {
        return rand();
    }

    int main(void)
    {
        int myarray[10];
        populate_array(myarray, 10, getNextRandomValue);
        ...
    }

```

Here, the *populate_array* function takes a function pointer as its third parameter, and calls it to get the values to populate the array with. We've written the callback *getNextRandomValue*, which returns a random-ish value, and passed a pointer to it to *populate_array*. *populate_array* function will call our callback function 10 times and assign the returned values to the elements in the given array.

This means, that a callback function is something which is passed as an argument to another function and is called when an event occurs inside the function or when some logic is processed and the function being passed is needed to be called back.

For the curious:

There are two ways that a callback is used: synchronous callback and asynchronous callback. A synchronous callback is provided to another function which is going to do some task and then return to the caller with the task completed. An asynchronous callback is provided to another function which is going to start a task and then return to the caller with the task possibly not completed.

A synchronous callback is typically used to provide a delegate to another function to which the other function delegates some step of the task. Classic examples of this delegation are the functions *bsearch()* and *qsort()* from the Standard C Library. Both of these functions take a callback which is used during the task the function is providing so that the type of the data being searched, in the case of *bsearch()*, or sorted, in the case of *qsort()*, does not need to be known by the function being used.

Making a Web Server in C

Now, we will start making a simple Web Server in C which can serve calls to GET endpoints in the program. Please note that the Onion library should be setup in order to use the mentioned program. The library can found at this link: <https://github.com/davidmoreno/onion>. The instructions to setup the library are very clearly given on the repository wiki.

We will start by providing a simple list of header libraries which we need to include in our program:

```

#include <onion/onion.h>
#include <onion/log.h>
#include <onion/version.h>

```

```
#include <signal.h>
#include <netdb.h>
#include <stdlib.h>
#include <unistd.h>
```

This include section involved libraries from the Onion library and some standard C libraries as well which help to call standard functions and use signal parameters from C binaries as well.

Even though it might look a little early but we will define two very simple callback functions now:

```
int hello(void *p, onion_request *req, onion_response *res){
    onion_response_write0(res,"pong");
    return OCS_PROCESSED;
}

onion *o = NULL;

static void shutdown_server(int _){
    if (o)
        onion_listen_stop(o);
}

onion_connection_status random_timeout(void *, onion_request *req, onion_response *res){
    int ms=2000 * ( ((float)RAND_MAX) / rand() );
    ONION_INFO("Wait %d ms", ms);
    usleep(ms * 1000);
    ONION_INFO("Done");
    onion_response_write(res,"OK",2);
    return OCS_PROCESSED;
}
```

Let us look at what we did in the above callback functions here:

- The first function we defined is *hello* function. This is a very simple function which has some arguments which includes Onion library's request and response reference object pointers. These pointers should be present in each and every callback function to allow them to modify response object and obtain the parameters present in the request object itself.
- This function simply writes Hello World string to the Onion library's response object. Please note that even though this function returns an integer and not the response object itself, the response object is still processed and is returned to the user when the integer returned is actually **just a signal to the system that method processing for HTTP call is complete**.
- The same integer is returned in the next callback function as well, *random_timeout*. This is done for the same reason.
- The second callback function just times out randomly. It first finds a random number, sleeps for some milliseconds, write a String in the response object and finally signals that the processing by the callback function is done.

Now, we finally need a main function to call everything of above we wrote. But wait, where did we define that which of these callback methods will be associated with what REST endpoint? Easily guessed, this will be done in the main function as well. Let us write down the main function for the program to do all of this:

```
int main(int argc, char **argv){
    signal(SIGINT, shutdown_server);
    signal(SIGTERM, shutdown_server);
```

```

ONION_VERSION_IS_COMPATIBLE_OR_ABORT();

o=onion_new(O_POOL);
onion_set_timeout(o, 5000);
onion_set_hostname(o,"0.0.0.0");
onion_url *urls=onion_root_url(o);

onion_url_add_static(urls, "static", "Hello static world", HTTP_OK);
onion_url_add(urls, "timeout", random_timeout);
onion_url_add(urls, "ping", hello);

onion_listen(o);
onion_free(o);
return 0;
}

```

We did many things in the function. Let us look at the points here:

- We provided signal parameters to the program with which server should be shut down. This is important to do to provide a way to gracefully shut down the web server without any unintended side-effects.
- Next, we check if the installed binaries on the platform are compatible with the Onion library version we are using in the program. If not, the program will abort.
- We then set some connection parameters like a timeout of 5 seconds (5000 milliseconds) and a hostname of 0.0.0.0 as well, which will allow HTTP calls to be done from anywhere to the server running this.
- We finally added some URLs. The first endpoint is a static endpoint with no callback method associated and it just returns a String and a 200 OK status code in response headers.
- The *timeout* endpoint is a GET endpoint, which when called will make a call to the callback function, *random_timeout*.
- The third URL we used is *ping* which will make a call to the callback function, *hello*.
- We finally started Onion to start listening using the connection parameters we defined.

Once you have Onion library setup, this program can be simply run using the command:

```
c server.c
```

Once you hit <http://localhost:8080/static>, we will see the static endpoint being called. Just like that, we can call other endpoints as well.

Conclusion

Onion is a very simple C library which can be used to make a simple, yet powerful Web Server with which you can server real-time and original web content for your web application. If you want to use it in a production environment, we will suggest to look for alternatives as well because of the sophistication of other programming languages and frameworks which are primarily defined and designed to server as Web Servers.

Even though there exist more frameworks which can be used to make much more sophisticated web services, C is still a powerful language for small projects and for quick setup.