Lab Report 1: CoAP Client

For this lab, a CoAP Client was developed. It was tested using coap.me public test server. The following features were implemented: **POST/PUT/GET/DELETE.**

Explanation:

Every method message is defined in a function, with only a few parameters changing for every method, for example for the GET method:

```
string get(string path) {
    // Declaring parameters
    string message = "";
    // Get the path length
    int size = path.length();

    // Declaring parameters as bits
    // Settings represent Version (01), Type (01) and Token length (0000)
    unsigned char settings = 0b01010000;
    // Method respresent the method used (GET = 0.01)
    unsigned char method = 0b00000001;
    // Generate a random message ID
    string msgId = randomMsgId();
    // uriOption contains the parameters for the host (3 = 0011 for uri-host and 7 = 0111 for size of coap.me)
    unsigned char uriOption = 0b00110111;
    // uri is coap.me in binary
    unsigned char uri[] = {0b01100011,0b01101111,0b01100001,0b01110000,0b00101110,0b01101101,0b01100101};
    // pathOption contains the parameters for the path (8 = 1000 for Location-path and size is defined with the switch)
    unsigned char pathOption = pathOptions(size);

    // Forming a message based on the parameters
    message.push_back(settings);
    message.push_back(method);
    message += msgId;
    message.push_back(uriOption);
    for (int i = 0; i < 7; i++){
        message.push_back(uri[i]);
    }
    message.push_back(pathOption);
    message += path;

    return message;
}
```

Here, all the binary parameters for the message are defined

After that, the message is created by concatenating the elements

The message is then sent using a UDP socket and a response is recovered:

```
// Function used to send requests to send a request to a coap server
void sendRequest(int sockfd, sockaddr_in servaddr, string message, char* buffer){
    // Size of the response
    unsigned int n = 0;
    // Size of the server address
    unsigned int len = 0;

    // Sending the message to the test server
    sendto(sockfd, message.c_str(), message.length(), MSG_CONFIRM, (const struct sockaddr *) &servaddr, sizeof(servaddr));

    // Get the response from the server
    n = recvfrom(sockfd, (char *)buffer, MAXLINE + 1,
            MSG_WAITALL, (struct sockaddr *) &servaddr,
            &len);
    // Closing the answer at the end
    buffer[n] = '\0';

    // Printing the headers and contents
    //cout << getHeaders(buffer) << endl;
    cout << getContent(buffer) << endl;
}
```

This response is parsed using the payload delimiter (11111111 in binary) to get the header but most importantly the contents of the response.

The main program is using a continuous loop with a menu that lets a user choose the method.

```
*------- Menu -------*
  1. Send a GET
  2. Send a POST
  3. Send a PUT
  4. Send a DELETE
  0. Quit
*--------------------*
Make your choice :
```

When an option is chosen (POST for example), the program asks for more information to send the request and displays the status of the request. For POST and PUT, a get is done immediately after to confirm the value was correctly registered in the path.

```
Enter the value you want to send : hello
Enter the path : sink
Status : POST OK
Contents : you put here: 23, and you put here: payload, and you put here: hello
```

We can also look at the packets in Wireshark to get more details and see that they are correctly formed:

```
▾ Constrained Application Protocol, Non-Confirmable, POST, MID:21695
    01.. .... = Version: 1
    ..01 .... = Type: Non-Confirmable (1)
    .... 0000 = Token Length: 0
    Code: POST (2)
    Message ID: 21695
  ▸ Opt Name: #1: Uri-Host: coap.me
  ▸ Opt Name: #2: Uri-Path: sink
  ▸ Opt Name: #3: Content-Format: text/plain; charset=utf-8
    End of options marker: 255
  ▸ Payload: Payload Content-Format: text/plain; charset=utf-8, Length: 5
    [Uri-Path: coap://coap.me/sink]
▾ Line-based text data: text/plain (1 lines)
    hello
```

Improvements:

A few nice to have features that were not mandatory for this project would be:

- To support long path names such as: .well-known/core;
- To let the user chose the URL or IP address of the server he wants to connect to;
- To implement block transfer to receive long messages;