Project GitHub link: https://github.com/AdelMirsharji/Token-generator-service

Token Generator Service

This document is not an educational document thus there may not be an explanation of the technologies used in this project because the sole purpose of this document is to explain the step-by-step implementation of the token generator service as explained below.

Token generator service is composed of three main parts:

1-Token generator server (in C).

2-A native linux library (in C++).

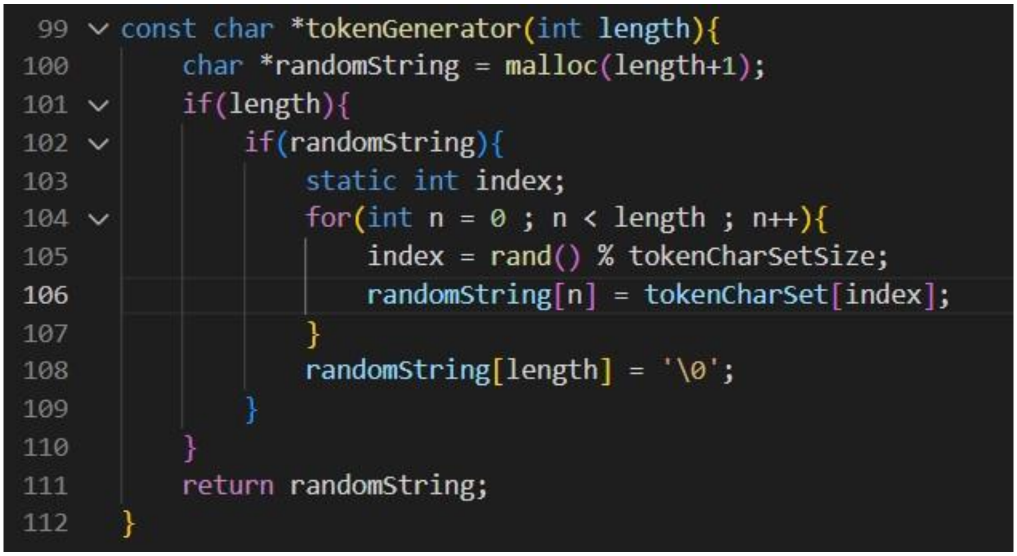
3-A java cli program that utilizes java native interface (in Java).

First let's start by explaining implementation of Token generator service (TGS). TGS contains two main parts:

1-Token generating using a random string generator function.

2-TCP server using socket programming.

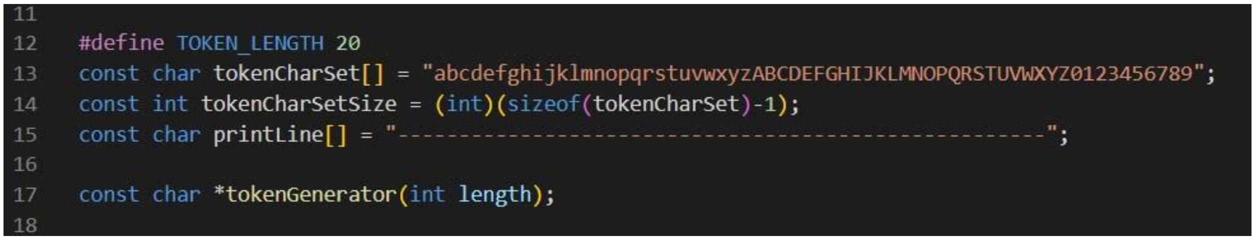
To create or generate tokens a function named "tokenGenerator" is implemented as below:



```
const char *tokenGenerator(int length){
    char *randomString = malloc(length+1);
    if(length){
        if(randomString){
            static int index;
            for(int n = 0 ; n < length ; n++){
                index = rand() % tokenCharSetSize;
                randomString[n] = tokenCharSet[index];
            }
            randomString[length] = '\0';
        }
    }
    return randomString;
}
```

In this function first we allocate memory for our randomString or token using malloc function in C. After that if considered length for token is more than 0 and randomString pointer is not NULL and memory allocation was done successfully there be a for loop to the size of considered token length that in it's body each character of token will be picked form a specific character set considered for the token and finally token string will be appended by a “\0” indicating end of the token string and at the end of the function token string will be returned.

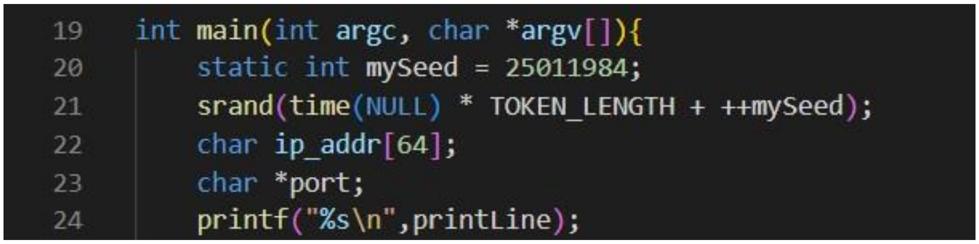
Now let’s see how token size and character set was implemented:



```c

#define TOKEN_LENGTH 20
const char tokenCharSet[] = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
const int tokenCharSetSize = (int)(sizeof(tokenCharSet)-1);
const char printLine[] = "-----------------------------------------------------";

const char *tokenGenerator(int length);

```

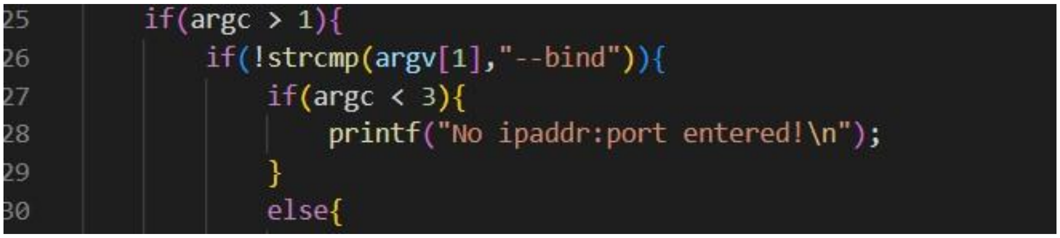
As we can see token length was defined by creating a macro using “#define” and token character set is a global const character array that contains all English alphabet in lower and upper case and numbers 0 to 9.

It’s notable that rand function is seeded as below (only once) at the start of main function:

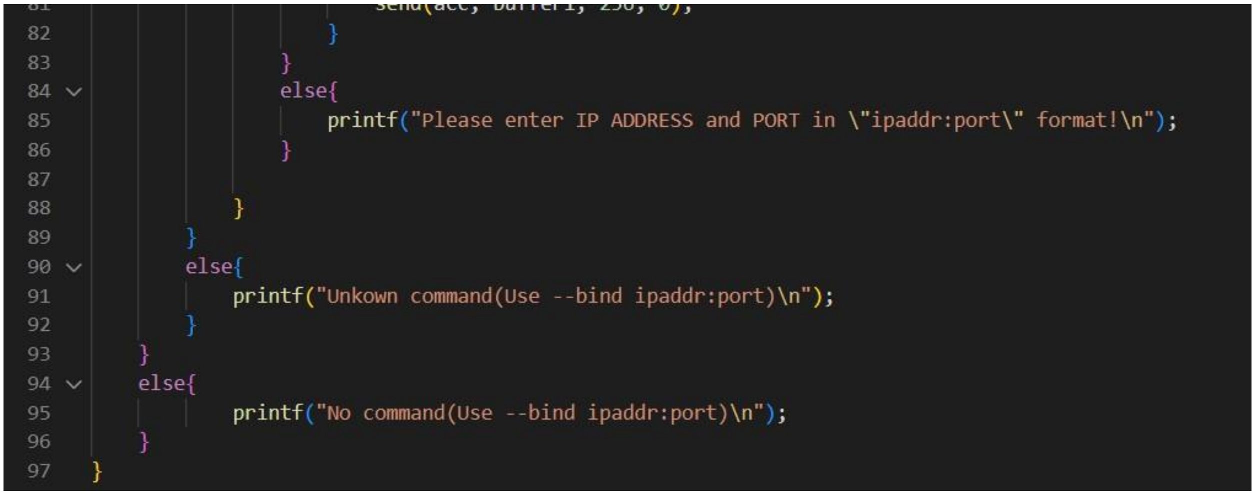


```c
int main(int argc, char *argv[]){
    static int mySeed = 25011984;
    srand(time(NULL) * TOKEN_LENGTH + ++mySeed);
    char ip_addr[64];
    char *port;
    printf("%s\n",printLine);
```

As for the second part of server source code in the main function first it’s checked that the inputs that program accepts are in true format or not because only one command can be given to server program and that is “--bind [ip:port]”.

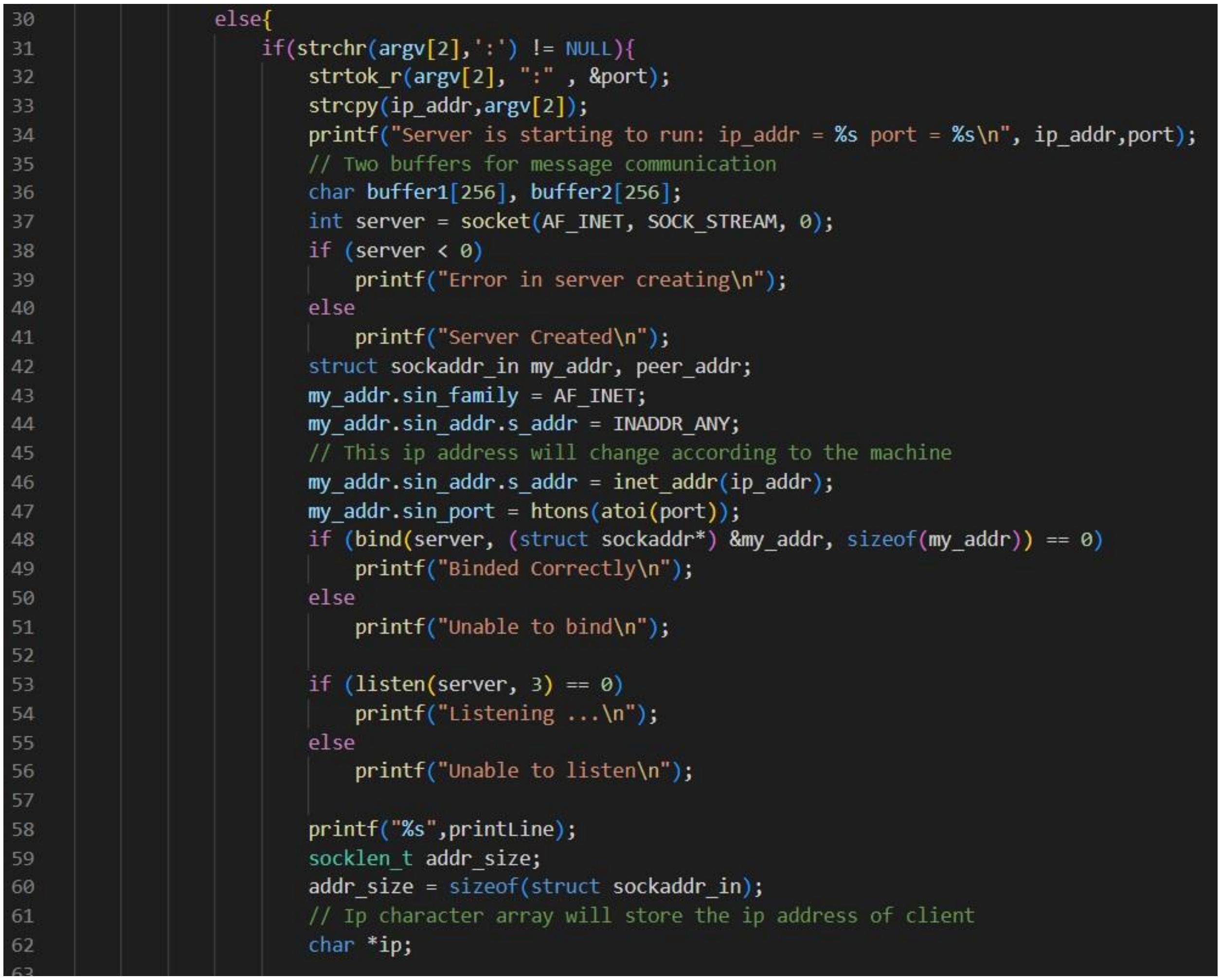


```c
    if(argc > 1){
        if(!strcmp(argv[1],"--bind")){
            if(argc < 3){
                printf("No ipaddr:port entered!\n");
            }
            else{
```



```
                        send(acc, buffer1, 256, 0);
                    }
                }
                else{
                    printf("Please enter IP ADDRESS and PORT in \"ipaddr:port\" format!\n");
                }

            }
        }
        else{
            printf("Unkown command(Use --bind ipaddr:port)\n");
        }
    }
    else{
            printf("No command(Use --bind ipaddr:port)\n");
    }
}
```

So if the argument passed to the server program in the cli is correct we proceed as follows.



```
            else{
                if(strchr(argv[2],':') != NULL){
                    strtok_r(argv[2], ":" , &port);
                    strcpy(ip_addr,argv[2]);
                    printf("Server is starting to run: ip_addr = %s port = %s\n", ip_addr,port);
                    // Two buffers for message communication
                    char buffer1[256], buffer2[256];
                    int server = socket(AF_INET, SOCK_STREAM, 0);
                    if (server < 0)
                        printf("Error in server creating\n");
                    else
                        printf("Server Created\n");
                    struct sockaddr_in my_addr, peer_addr;
                    my_addr.sin_family = AF_INET;
                    my_addr.sin_addr.s_addr = INADDR_ANY;
                    // This ip address will change according to the machine
                    my_addr.sin_addr.s_addr = inet_addr(ip_addr);
                    my_addr.sin_port = htons(atoi(port));
                    if (bind(server, (struct sockaddr*) &my_addr, sizeof(my_addr)) == 0)
                        printf("Binded Correctly\n");
                    else
                        printf("Unable to bind\n");

                    if (listen(server, 3) == 0)
                        printf("Listening ...\n");
                    else
                        printf("Unable to listen\n");

                    printf("%s",printLine);
                    socklen_t addr_size;
                    addr_size = sizeof(struct sockaddr_in);
                    // Ip character array will store the ip address of client
                    char *ip;
```

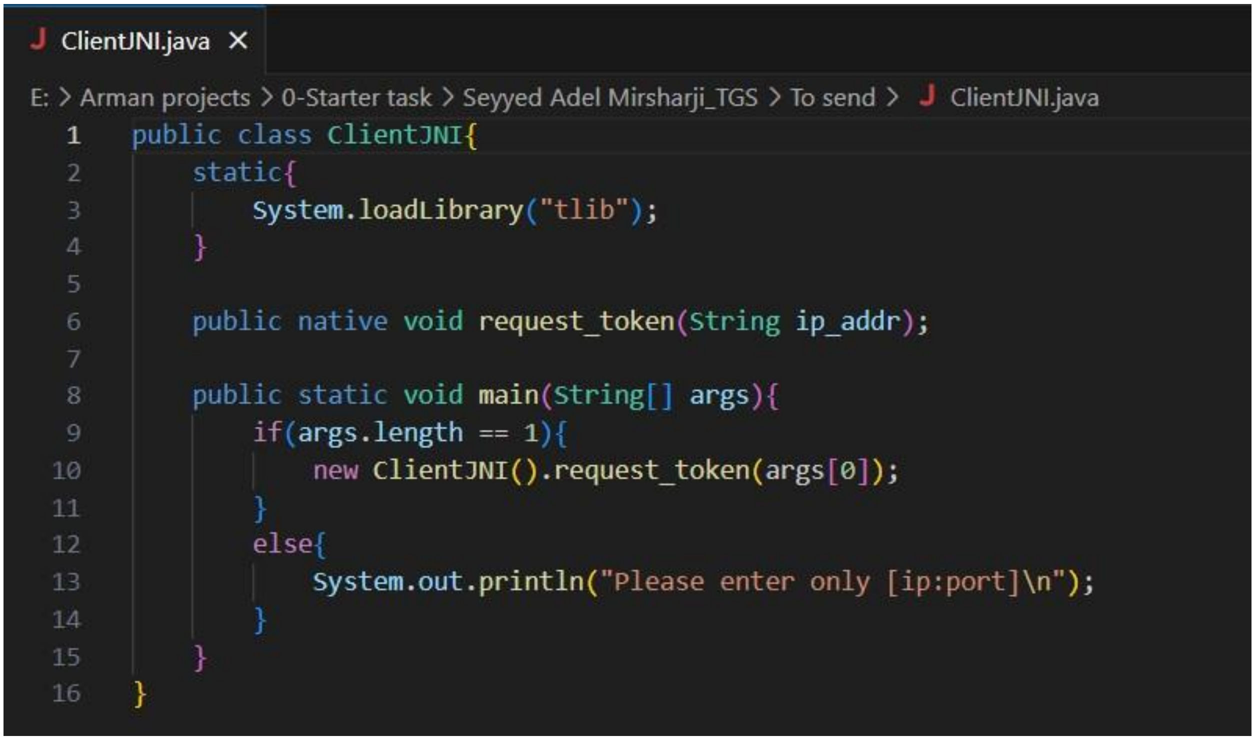
First [ip:port] input string is spited into two parts, ip and port using “strtok_r” function.

After that there are some basic socket programming configurations in order to be able to create server and bind it to the specific ip and port and then start listening.

Afterwards there is a forever while loop to be able to accept connections of clients in future one by one.

So now we have “tserver.c” ready to be compiled into an executable file.

The next two parts native library and jni will be implemented in parallel. At first a we declare a function with the native keyboard.


ClientJNI.java ×
E: > Arman projects > 0-Starter task > Seyyed Adel Mirsharji_TGS > To send > ClientJNI.java

```
public class ClientJNI{
    static{
        System.loadLibrary("tlib");
    }

    public native void request_token(String ip_addr);

    public static void main(String[] args){
        if(args.length == 1){
            new ClientJNI().request_token(args[0]);
        }
        else{
            System.out.println("Please enter only [ip:port]\n");
        }
    }
}
```

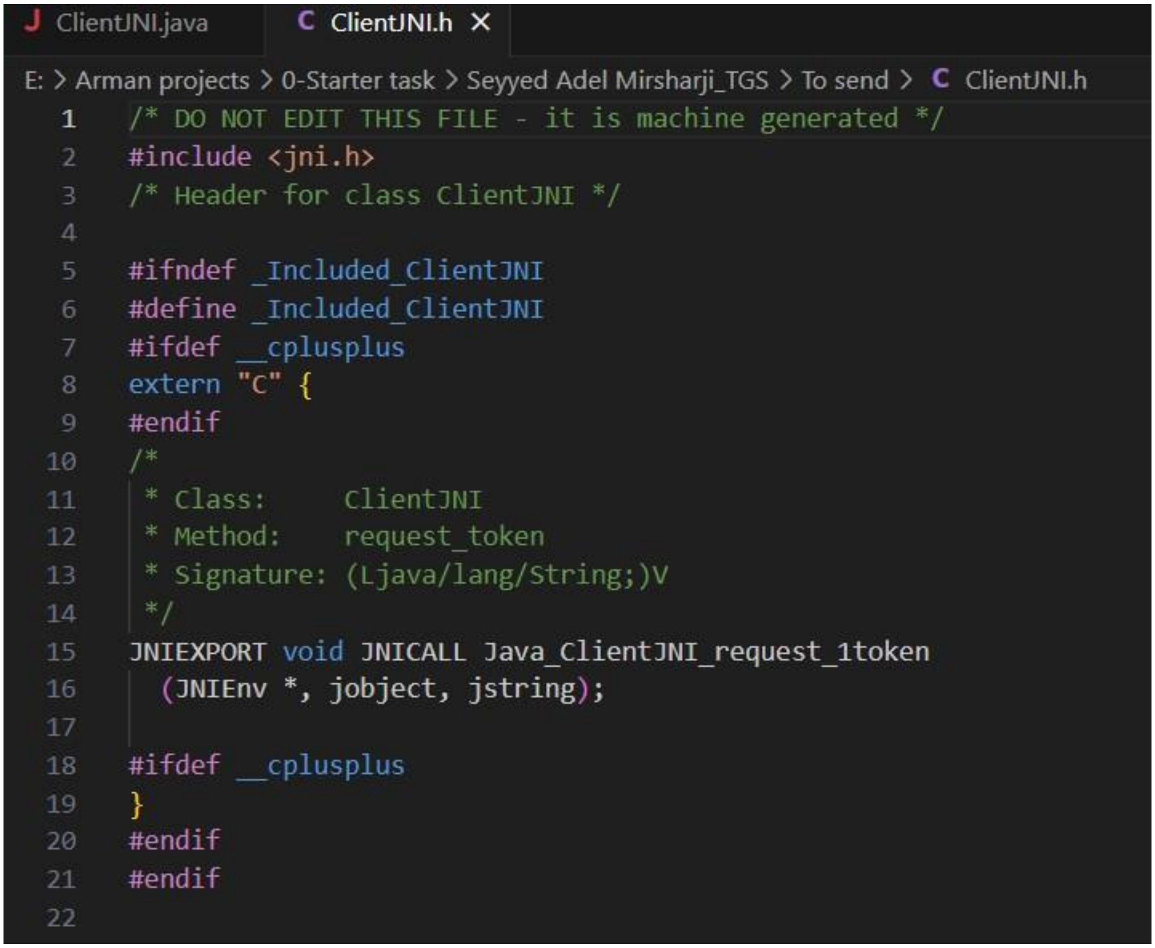
In first block we are loading a so library called “tlib”. The reason why it’s inside a static block is we need to load the library as soon as our class is loaded. Next a native function “request_token” that accepts string as input is declared. In the main function it is checked that if needed argument “[ip:port]” is passed to jni in cli or not.

Next step is to create a header file that has all the native function declarations that we’ve declared in the java file.

In order to do so we should run the command below:

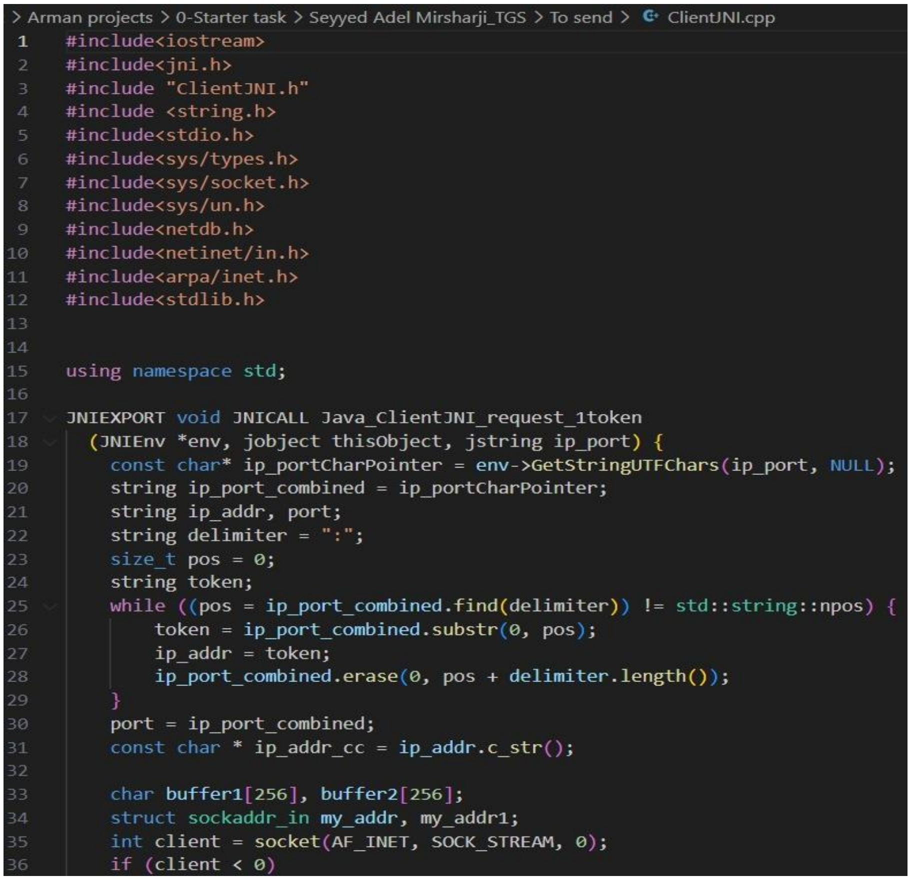
```
$ javac -h . ClientJNI.java
```

It will generate a header file named “ClientJNI.h” which declares a single function with the same signature (in JNI-Style) as request_token().


ClientJNI.java
ClientJNI.h
E: > Arman projects > 0-Starter task > Seyyed Adel Mirsharji_TGS > To send > ClientJNI.h

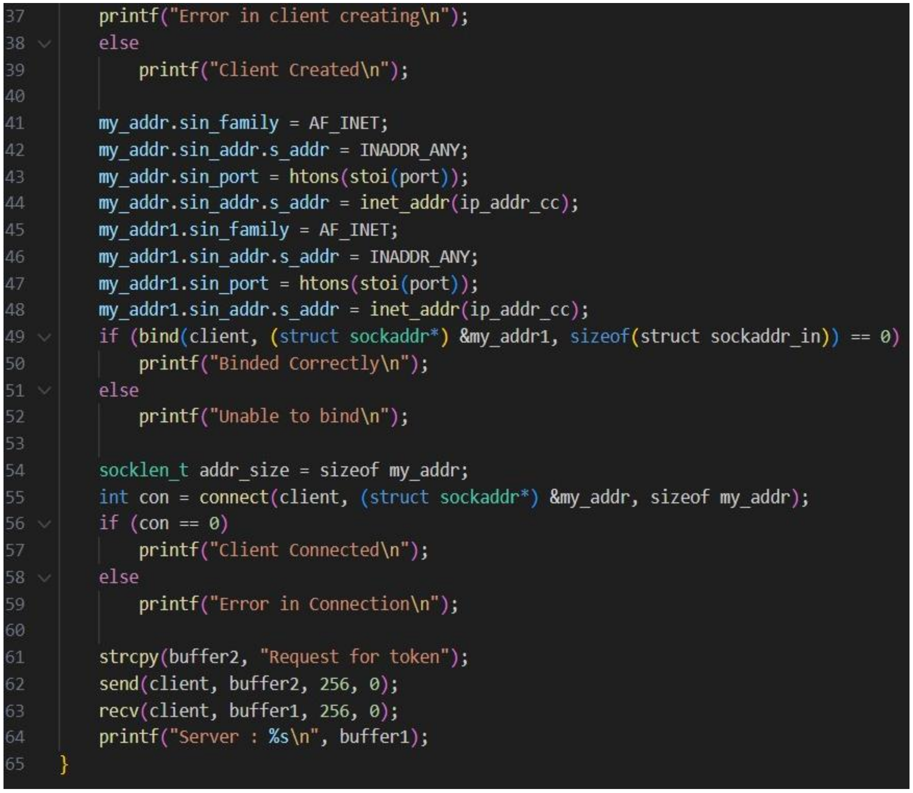
```c
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class ClientJNI */

#ifndef _Included_ClientJNI
#define _Included_ClientJNI
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     ClientJNI
 * Method:    request_token
 * Signature: (Ljava/lang/String;)V
 */
JNIEXPORT void JNICALL Java_ClientJNI_request_1token
  (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif

```

Next step is to Create a CPP file and generate an Object file from it. Now we are going to create a C++ file that has a function with the same method signature as in “ClientJNI.h”.



> Arman projects > 0-Starter task > Seyyed Adel Mirsharji_TGS > To send > ClientJNI.cpp

```
#include<iostream>
#include<jni.h>
#include "ClientJNI.h"
#include <string.h>
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<netdb.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>


using namespace std;

JNIEXPORT void JNICALL Java_ClientJNI_request_1token
  (JNIEnv *env, jobject thisObject, jstring ip_port) {
    const char* ip_portCharPointer = env->GetStringUTFChars(ip_port, NULL);
    string ip_port_combined = ip_portCharPointer;
    string ip_addr, port;
    string delimiter = ":";
    size_t pos = 0;
    string token;
    while ((pos = ip_port_combined.find(delimiter)) != std::string::npos) {
        token = ip_port_combined.substr(0, pos);
        ip_addr = token;
        ip_port_combined.erase(0, pos + delimiter.length());
    }
    port = ip_port_combined;
    const char * ip_addr_cc = ip_addr.c_str();

    char buffer1[256], buffer2[256];
    struct sockaddr_in my_addr, my_addr1;
    int client = socket(AF_INET, SOCK_STREAM, 0);
    if (client < 0)
```



```
    printf("Error in client creating\n");
    else
        printf("Client Created\n");

    my_addr.sin_family = AF_INET;
    my_addr.sin_addr.s_addr = INADDR_ANY;
    my_addr.sin_port = htons(stoi(port));
    my_addr.sin_addr.s_addr = inet_addr(ip_addr_cc);
    my_addr1.sin_family = AF_INET;
    my_addr1.sin_addr.s_addr = INADDR_ANY;
    my_addr1.sin_port = htons(stoi(port));
    my_addr1.sin_addr.s_addr = inet_addr(ip_addr_cc);
    if (bind(client, (struct sockaddr*) &my_addr1, sizeof(struct sockaddr_in)) == 0)
        printf("Binded Correctly\n");
    else
        printf("Unable to bind\n");

    socklen_t addr_size = sizeof my_addr;
    int con = connect(client, (struct sockaddr*) &my_addr, sizeof my_addr);
    if (con == 0)
        printf("Client Connected\n");
    else
        printf("Error in Connection\n");

    strcpy(buffer2, "Request for token");
    send(client, buffer2, 256, 0);
    recv(client, buffer1, 256, 0);
    printf("Server : %s\n", buffer1);
}
```

In the client side we do some basic socket configuration to be able to connect to our server. Pay attention that ip address and port number are received form java cli program and if they match the ones that server is listening on the connection will be successful and client will receive a unique token from server.

Next if we installed java JDK correctly we can see the path using commands below:

```
. /etc/environment
echo $JAVA_HOME
```

Note that if there is nothing printed as the java home pass you should set it in your system environment. (for further instructions search how to set environment variable in linux using nano editor).

Now we will run the command below to create an object file:

```
g++ -c -fPIC -I${JAVA_HOME}/include -I${JAVA_HOME}/include/linux
ClientJNI.cpp -o ClientJNI.o
```

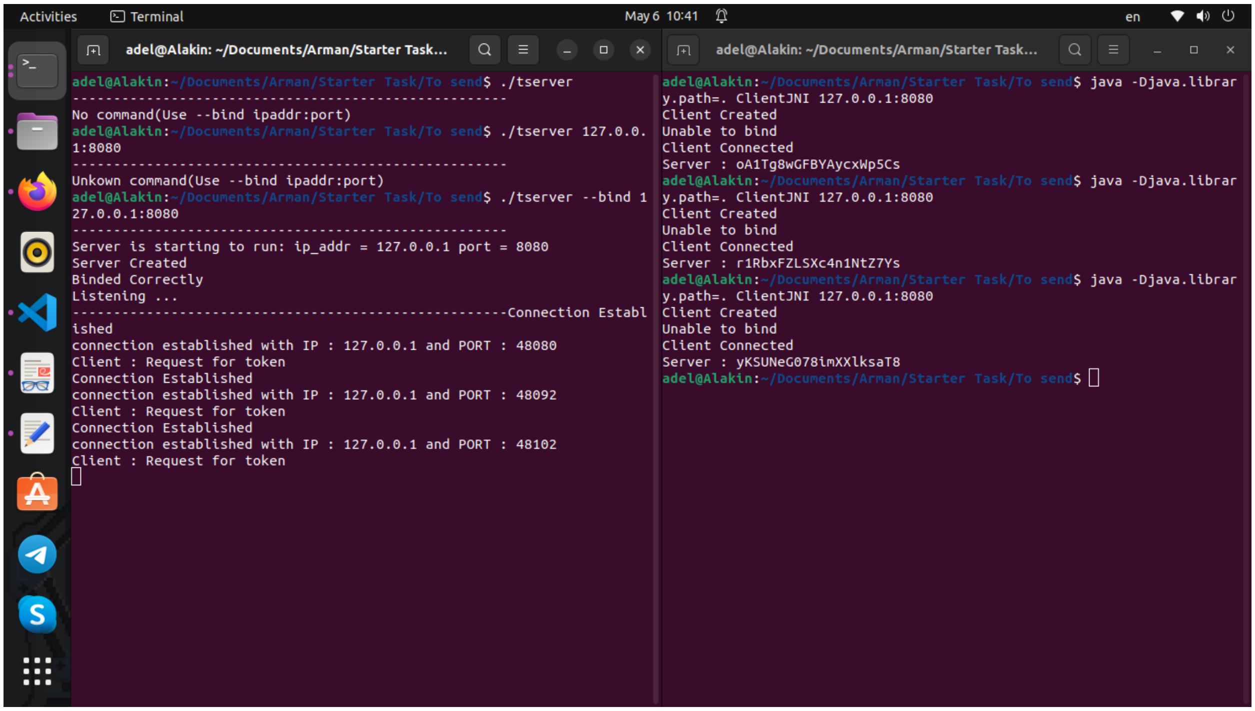
This will create an object file named “ClientJNI.o”.Next we will create a .so file from this object file running the command below:

```
gcc -fPIC -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared
-o libtlib.so ClientJNI.cpp
```

Now we can run our java program using the command below:

```
java -Djava.library.path=. ClientJNI 127.0.0.1:8080
```

This will result in something like below picture: (note that first you should run tserver executable)


Activities
Terminal
May 6 10:41
en
adel@Alakin: ~/Documents/Arman/Starter Task...
adel@Alakin: ~/Documents/Arman/Starter Task...

```
adel@Alakin:~/Documents/Arman/Starter Task/To send$ ./tserver
-----------------------------------------------------
No command(Use --bind ipaddr:port)
adel@Alakin:~/Documents/Arman/Starter Task/To send$ ./tserver 127.0.0.
1:8080
-----------------------------------------------------
Unkown command(Use --bind ipaddr:port)
adel@Alakin:~/Documents/Arman/Starter Task/To send$ ./tserver --bind 1
27.0.0.1:8080
-----------------------------------------------------
Server is starting to run: ip_addr = 127.0.0.1 port = 8080
Server Created
Binded Correctly
Listening ...
-----------------------------------------------------Connection Establ
ished
connection established with IP : 127.0.0.1 and PORT : 48080
Client : Request for token
Connection Established
connection established with IP : 127.0.0.1 and PORT : 48092
Client : Request for token
Connection Established
connection established with IP : 127.0.0.1 and PORT : 48102
Client : Request for token
```

```
adel@Alakin:~/Documents/Arman/Starter Task/To send$ java -Djava.librar
y.path=. ClientJNI 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : oA1Tg8wGFBYAycxWp5Cs
adel@Alakin:~/Documents/Arman/Starter Task/To send$ java -Djava.librar
y.path=. ClientJNI 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : r1RbxFZLSXc4n1NtZ7Ys
adel@Alakin:~/Documents/Arman/Starter Task/To send$ java -Djava.librar
y.path=. ClientJNI 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : yKSUNeG078imXXlksaT8
adel@Alakin:~/Documents/Arman/Starter Task/To send$
```

As we can see multiple clients received different tokens from server and connection was successful. It is also visible that entering wrong commands will result in some what specific errors.

Now we can take one step further and create a jar file from out java file. In order to do so we should create a directory named “build” in the same directory as java code and run the commands below:

```
javac -d ./build ClientJNI.java
cd build
touch MANIFEST.MF
```

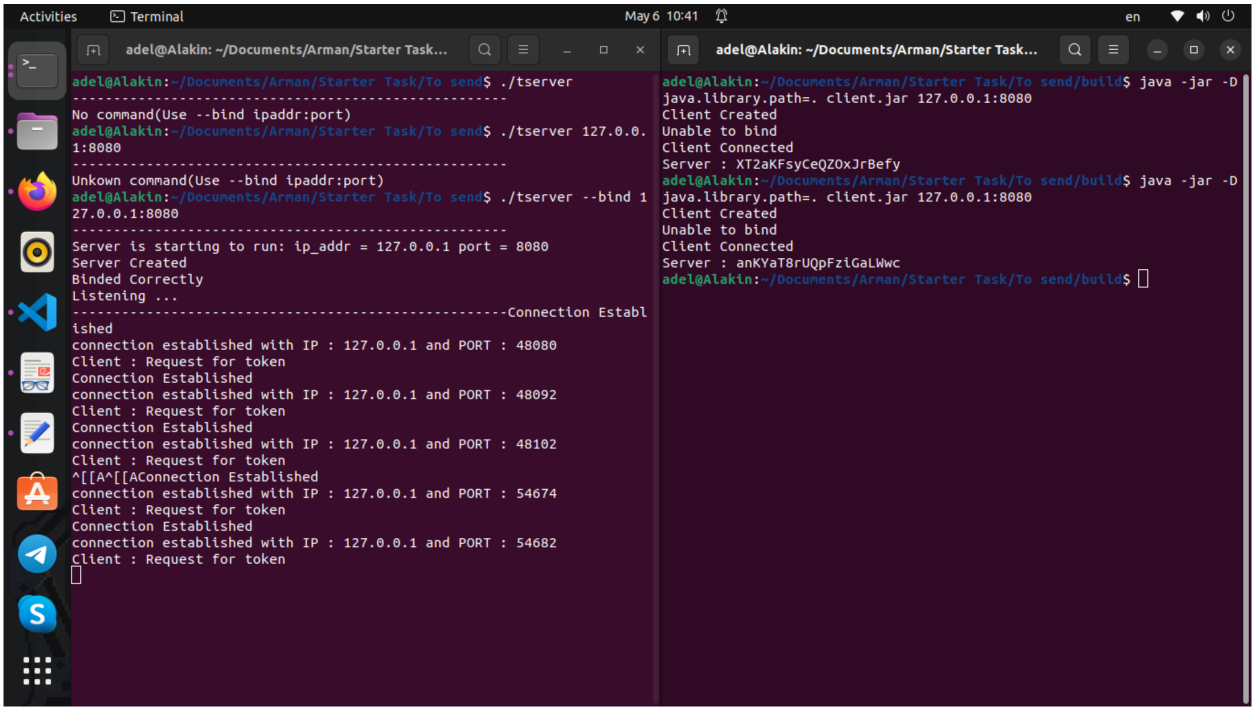
The final command will create a manifest file in the build directory (it is needed to make jar file executable). After it is created open the file and enter one line in it:

Main-Class: ClientJNI

Next run the commands below to run jar file. (note that tlib library must be present in the same directory as jar file)

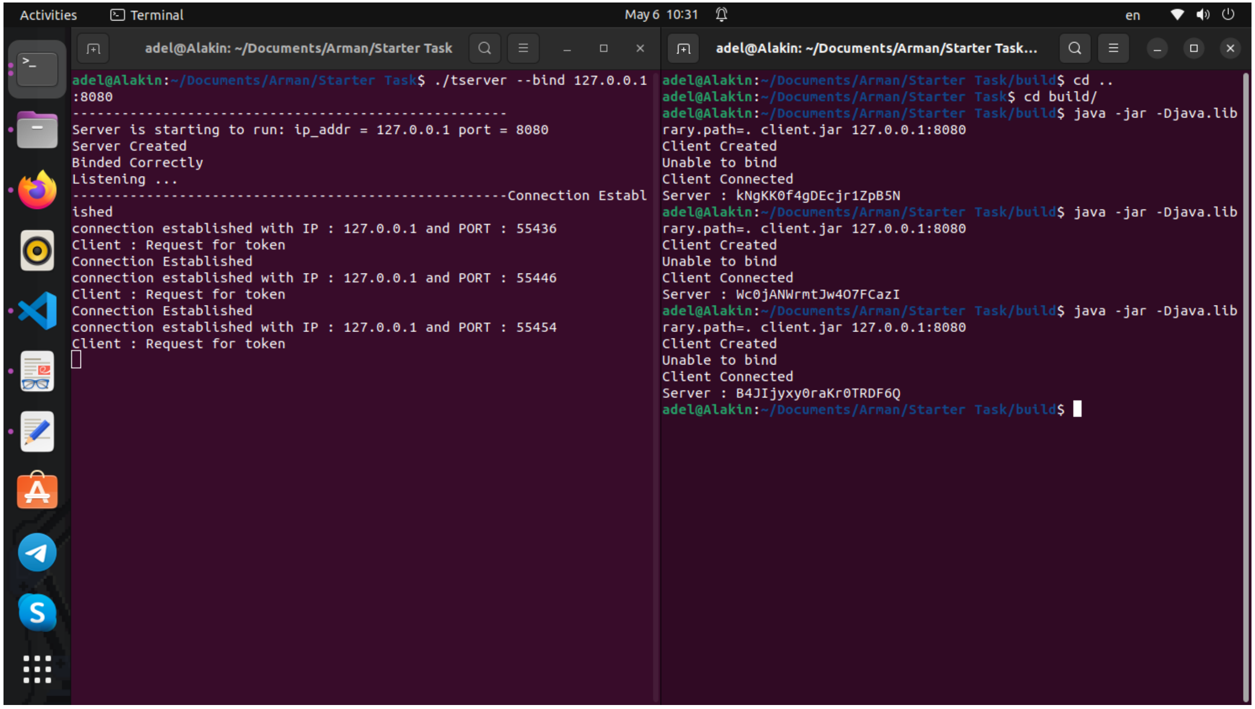
```
jar cmvf MANIFEST.MF client.jar ClientJNI.class
java -jar -Djava.library.path=. client.jar
```

This will result in something like pictures below:


Activities
Terminal
May 6 10:41
en
adel@Alakin: ~/Documents/Arman/Starter Task...
adel@Alakin: ~/Documents/Arman/Starter Task...

```
adel@Alakin:~/Documents/Arman/Starter Task/To send$ ./tserver
-----------------------------------------------------
No command(Use --bind ipaddr:port)
adel@Alakin:~/Documents/Arman/Starter Task/To send$ ./tserver 127.0.0.
1:8080
-----------------------------------------------------
Unkown command(Use --bind ipaddr:port)
adel@Alakin:~/Documents/Arman/Starter Task/To send$ ./tserver --bind 1
27.0.0.1:8080
-----------------------------------------------------
Server is starting to run: ip_addr = 127.0.0.1 port = 8080
Server Created
Binded Correctly
Listening ...
-----------------------------------------------------Connection Establ
ished
connection established with IP : 127.0.0.1 and PORT : 48080
Client : Request for token
Connection Established
connection established with IP : 127.0.0.1 and PORT : 48092
Client : Request for token
Connection Established
connection established with IP : 127.0.0.1 and PORT : 48102
Client : Request for token
^[[A^[[AConnection Established
connection established with IP : 127.0.0.1 and PORT : 54674
Client : Request for token
Connection Established
connection established with IP : 127.0.0.1 and PORT : 54682
Client : Request for token
```

```
adel@Alakin:~/Documents/Arman/Starter Task/To send/build$ java -jar -D
java.library.path=. client.jar 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : XT2aKFsyCeQZOxJrBefy
adel@Alakin:~/Documents/Arman/Starter Task/To send/build$ java -jar -D
java.library.path=. client.jar 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : anKYaT8rUQpFziGaLWwc
adel@Alakin:~/Documents/Arman/Starter Task/To send/build$
```


Activities
Terminal
May 6 10:31
en
adel@Alakin: ~/Documents/Arman/Starter Task
adel@Alakin: ~/Documents/Arman/Starter Task...

```
adel@Alakin:~/Documents/Arman/Starter Task$ ./tserver --bind 127.0.0.1
:8080
-----------------------------------------------------
Server is starting to run: ip_addr = 127.0.0.1 port = 8080
Server Created
Binded Correctly
Listening ...
-----------------------------------------------------Connection Establ
ished
connection established with IP : 127.0.0.1 and PORT : 55436
Client : Request for token
Connection Established
connection established with IP : 127.0.0.1 and PORT : 55446
Client : Request for token
Connection Established
connection established with IP : 127.0.0.1 and PORT : 55454
Client : Request for token
```

```
adel@Alakin:~/Documents/Arman/Starter Task/build$ cd ..
adel@Alakin:~/Documents/Arman/Starter Task$ cd build/
adel@Alakin:~/Documents/Arman/Starter Task/build$ java -jar -Djava.lib
rary.path=. client.jar 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : kNgKK0f4gDEcjr1ZpB5N
adel@Alakin:~/Documents/Arman/Starter Task/build$ java -jar -Djava.lib
rary.path=. client.jar 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : Wc0jANWrmtJw4O7FCazI
adel@Alakin:~/Documents/Arman/Starter Task/build$ java -jar -Djava.lib
rary.path=. client.jar 127.0.0.1:8080
Client Created
Unable to bind
Client Connected
Server : B4JIjyxy0raKr0TRDF6Q
adel@Alakin:~/Documents/Arman/Starter Task/build$
```