

Учреждение образования "Полоцкий Государственный Университет"

Факультет информационных технологий  
Кафедра вычислительных систем и сетей

## **ЛАБОРАТОРНАЯ РАБОТА № 2**

по дисциплине: **"Защита информационных ресурсов компьютерных  
систем и сетей"**

ВЫПОЛНИЛ

студент группы 16-ИТ-3  
Яблонский А. С.  
вариант № 7

ПРОВЕРИЛ

преподаватель  
Ярошевич П.В.

Полоцк 2019 г.

# 1 Задача

В рамках данной лабораторной работы необходимо ознакомиться и изучить функции библиотеки Open SSL, а также создать приложение, которое согласно варианту задания, которое использует защищенное соединение.

## 2 Вариант № 7

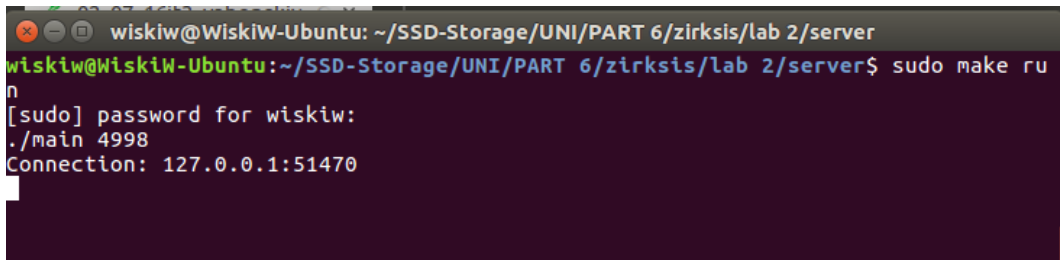
В рамках данной лабораторной работы необходимо реализовать взаимодействие типа клиент - сервер. Клиент делает запрос серверу на выполнение какой-либо команды. Сервер выполняет эту команду и возвращает результаты клиенту. Протокол ТСР. Поддержка нескольких клиентов одновременно.

## 3 Ход выполнения

Выполнение лабораторной работы включало в себя следующие шаги:

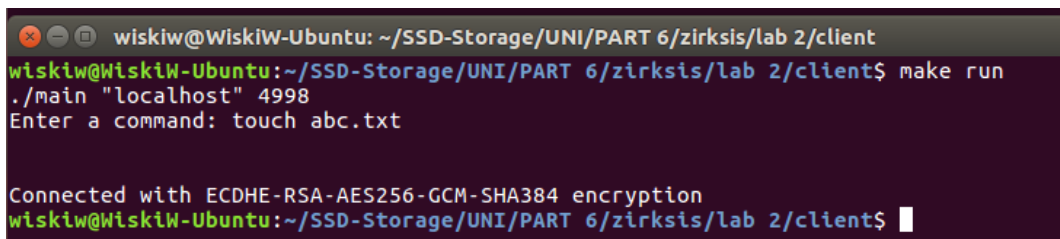
1. Создание сервера.
2. Создание клиента.
3. Установка соединений.
4. Реализация функции выполнения сервером консольных команд с клиента.
5. Реализация возможности одновременного соединения нескольких клиентов.
6. Компиляция и запуск программы.

## 4 Скриншоты



```
wiskiW@WiskiW-Ubuntu: ~/SSD-Storage/UNI/PART 6/zirksis/lab 2/server
wiskiW@WiskiW-Ubuntu:~/SSD-Storage/UNI/PART 6/zirksis/lab 2/server$ sudo make ru
n
[sudo] password for wiskiW:
./main 4998
Connection: 127.0.0.1:51470
```

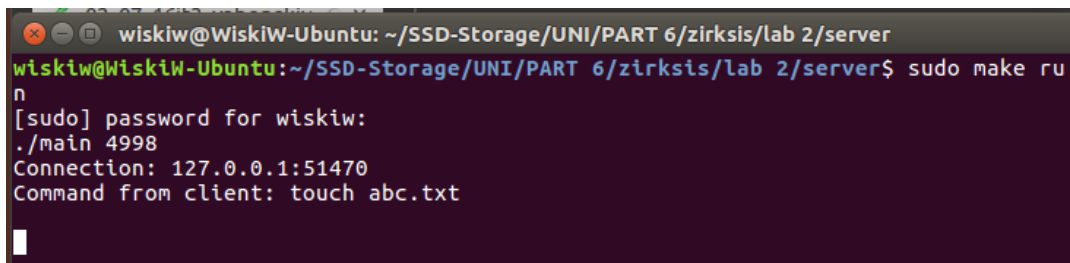
Рис. 1: Установка соединения серверной части



```
wiskiW@WiskiW-Ubuntu: ~/SSD-Storage/UNI/PART 6/zirksis/lab 2/client
wiskiW@WiskiW-Ubuntu:~/SSD-Storage/UNI/PART 6/zirksis/lab 2/client$ make run
./main "localhost" 4998
Enter a command: touch abc.txt

Connected with ECDHE-RSA-AES256-GCM-SHA384 encryption
wiskiW@WiskiW-Ubuntu:~/SSD-Storage/UNI/PART 6/zirksis/lab 2/client$
```

Рис. 2: Установка соединения клиентской части



```
wiskiW@WiskiW-Ubuntu: ~/SSD-Storage/UNI/PART 6/zirksis/lab 2/server
wiskiW@WiskiW-Ubuntu:~/SSD-Storage/UNI/PART 6/zirksis/lab 2/server$ sudo make ru
n
[sudo] password for wiskiW:
./main 4998
Connection: 127.0.0.1:51470
Command from client: touch abc.txt
```

Рис. 3: Результат работы программы

## 5 Source Code

### client.c

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <sys/socket.h>
#include <resolv.h>
#include <netdb.h>
#include <openssl/ssl.h>
#include <openssl/err.h>

#define FAIL    -1

int openConnection(const char *hostname, int port){
    int sd;
    struct hostent *host;
    struct sockaddr_in addr;

    if ( (host = gethostbyname(hostname)) == NULL ){
        perror(hostname);
        abort();
    }
    sd = socket(PF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = *(long*)(host->h_addr);
    if ( connect(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0 ){
        close(sd);
        perror(hostname);
        abort();
    }
    return sd;
}

SSL_CTX* initCTX(void){
```

```

SSL_METHOD *method;
SSL_CTX *ctx;

OpenSSL_add_all_algorithms();
SSL_load_error_strings();
method = TLSv1_2_client_method();
ctx = SSL_CTX_new(method);
if ( ctx == NULL ){
    ERR_print_errors_fp(stderr);
    abort();
}
return ctx;
}

void showCerts(SSL* ssl){
    X509 *cert;
    char *line;

    cert = SSL_get_peer_certificate(ssl);
    if ( cert != NULL ){
        line = X509_NAME_oneline(X509_get_subject_name(cert), 0, 0);
        free(line);
        line = X509_NAME_oneline(X509_get_issuer_name(cert), 0, 0);
        free(line);
        X509_free(cert);
    }
}

int main(int count, char *strings[]){
    SSL_CTX *ctx;
    int server;
    SSL *ssl;
    char buf[1024];
    int bytes;
    char *hostname, *portnum;

    if ( count != 3 ){

```

```

        printf("usage:_%s_<hostname>_<portnum>\n", strings[0]);
        exit(0);
    }
    SSL_library_init();
    hostname=strings[1];
    portnum=strings[2];

    ctx = initCTX();
    server = openConnection(hostname, atoi(portnum));
    ssl = SSL_new(ctx);
    SSL_set_fd(ssl, server);

    if ( SSL_connect(ssl) == FAIL ){
        ERR_print_errors_fp(stderr);

    } else {
        char command[256];
        printf("Enter_a_command:_");
        fgets(command, 255, stdin);

        printf("\n\nConnected_with_%s_encryption\n", SSL_get_cipher(
            ssl));
        showCerts(ssl);
        SSL_write(ssl, command, strlen(command));
        bytes = SSL_read(ssl, buf, sizeof(buf));
        buf[bytes] = 0;
        printf("%s", buf);
        SSL_free(ssl);
    }
    close(server);
    SSL_CTX_free(ctx);
    return 0;
}

```

#### server.c

```

#include <errno.h>
#include <unistd.h>
#include <malloc.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/socket.h>

```

```

#include <sys/types.h>
#include <netinet/in.h>
#include <resolv.h>
#include "openssl/ssl.h"
#include "openssl/err.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <dirent.h>
#include <pthread.h>

#define FAIL    -1

int openListener(int port){
    int sd;
    struct sockaddr_in addr;

    sd = socket(PF_INET, SOCK_STREAM, 0);
    bzero(&addr, sizeof(addr));
    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = INADDR_ANY;
    if (bind(sd, (struct sockaddr*)&addr, sizeof(addr)) != 0 ) {
        perror("Can't_bind_port");
        abort();
    }
    if ( listen(sd, 10) != 0 ) {
        perror("Can't_configure_listening_port");
        abort();
    }
    return sd;
}

int isRoot() {
    if (getuid() != 0) {
        return 0;
    }
    else {

```

```

        return 1;
    }

}

SSL_CTX* initServerCTX(void) {    SSL_METHOD *method;
    SSL_CTX *ctx;

    OpenSSL_add_all_algorithms();
    SSL_load_error_strings();
    method = TLSv1_2_server_method();
    ctx = SSL_CTX_new(method);
    if ( ctx == NULL ) {
        ERR_print_errors_fp(stderr);
        abort();
    }
    return ctx;
}

void loadCertificates(SSL_CTX* ctx, char* CertFile, char* KeyFile) {

    if ( SSL_CTX_use_certificate_file(ctx, CertFile, SSL_FILETYPE_PEM)
        ERR_print_errors_fp(stderr);
        abort();
    }
    if ( SSL_CTX_use_PrivateKey_file(ctx, KeyFile, SSL_FILETYPE_PEM)
        ERR_print_errors_fp(stderr);
        abort();
    }
    if ( !SSL_CTX_check_private_key(ctx) ) {
        fprintf(stderr, "Private_key_does_not_match_the_public_certificate\n");
        abort();
    }
}

void showCerts(SSL* ssl) {    X509 *cert;
    char *line;

    cert = SSL_get_peer_certificate(ssl);
    if ( cert != NULL ) {
        line = X509_NAME_oneline(X509_get_subject_name(cert), 0, 0);

```



```

        free(line);
        line = X509_NAME_oneline(X509_get_issuer_name(cert), 0, 0);
        free(line);
        X509_free(cert);
    }
}

void exec(const char* cmd) {
    char buffer[128];
    FILE* pipe = popen(cmd, "r");
    if (!pipe) {
        printf("popen()_failed!_%s\n", cmd);
    }
    pclose(pipe);
    //return result;
}

void* clientFlow(void* sslka) {
    SSL* ssl = (SSL*) sslka;
    char buf[1024] = {0};

    int sd, bytes;

    if ( SSL_accept(ssl) == FAIL )
        ERR_print_errors_fp(stderr);
    else {
        showCerts(ssl);
        bytes = SSL_read(ssl, buf, sizeof(buf));
        buf[bytes] = '\0';

        printf("Command_from_client:_%s\n", buf);

        if ( bytes > 0 ) {
            char str[1024] = "";
            char buff[64];

            exec(buf);
            SSL_write(ssl, str, sizeof(str));

        } else {

```

```

ERR_print_errors_fp(stderr);
    }
}
sd = SSL_get_fd(ssl);
SSL_free(ssl);
close(sd);
return NULL;
}

```

```

int main(int count, char *argv[]) {
    SSL_CTX *ctx;
    int server;
    char *portnum;
    pthread_t thread;

    if(!isRoot()) {
        printf("This_program_must_be_run_as_root/sudo_user!!");
        exit(0);
    }
    if ( count != 2 ) {
        printf("Usage:_%s_<portnum>\n", argv[0]);
        exit(0);
    }

    SSL_library_init();

    portnum = argv[1];
    ctx = initServerCTX();
    loadCertificates(ctx, "mycert.pem", "mycert.pem");
    server = openListener(atoi(portnum));
    while (1) {
        struct sockaddr_in addr;
        socklen_t len = sizeof(addr);
        SSL *ssl;

        int client = accept(server, (struct sockaddr*)&addr, &len);
        printf("Connection:_%s:%d\n", inet_ntoa(addr.sin_addr), ntohs

```

```

        ssl = SSL_new(ctx);
        SSL_set_fd(ssl, client);
        if(pthread_create(&thread, NULL, clientFlow, (void*) ssl) !=
            close(client);
    }
}
close(server);
SSL_CTX_free(ctx);
}

```