**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Операционные системы»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9**

«Сети»

Содержание

[Введение 3](#_Toc185672618)

[1 Сокеты tcp 4](#_Toc185672619)

[1.1 Задание 4](#_Toc185672620)

[1.2 Ход работы 4](#_Toc185672621)

[1.2.1 Теория 4](#_Toc185672622)

[1.2.2 Программа для сервера 5](#_Toc185672623)

[1.2.3 Программа для клиента 6](#_Toc185672624)

[1.2.4 Тестирование 9](#_Toc185672625)

[2 rpc-программа 12](#_Toc185672626)

[2.1 Задание 12](#_Toc185672627)

[2.2 Ход работы 12](#_Toc185672628)

[2.2.1 Теория 12](#_Toc185672629)

[2.2.2 Реализация программы 12](#_Toc185672630)

[Заключение **17**](#_Toc185672631)

[Список использованных источников 18](#_Toc185672632)

Введение

Цель работы – протестировать работу сокетов tcp при различных настройках setsockopt и реализовать rpc-программу для linux с поддержкой аутентификации.

Для достижения поставленной цели необходимо решить следующие задачи:

* написать программу tcp для сервера;
* написать программу tcp для клиента;
* протестировать работу сокетов tcp при разных настрйках;
* реализовать rpc-программу для linux с поддержкой аутентификации.

# Сокеты tcp

## Задание

Протестировать работу сокетов tcp при различных настройках setsockopt.

## Ход работы

### Теория

Сетевые протоколы — это набор правил и стандартов, которые определяют, как данные передаются и обрабатываются в компьютерных сетях. Они обеспечивают взаимодействие между различными устройствами и приложениями, позволяя им корректно обмениваться информацией.

**TCP-сокеты (Transmission Control Protocol) — это тип сокетов, который используется для установления надёжного и устойчивого соединения между клиентом и сервером.** TCP-сокеты обеспечивают проверку доставки данных, управление потоком данных и гарантированную доставку пакетов в правильном порядке.

setsockopt` — это функция в программировании на языке C (и других языках), которая позволяет настраивать параметры сокетов. С помощью этой функции можно изменить поведение сокета в зависимости от требований приложения.

Настройки, которые можно изменить с помощью `setsockopt`:

1. SO\_REUSEADDR: позволяет повторно использовать адрес и порт, даже если они находятся в состоянии TIME\_WAIT. Это полезно для перезапуска серверов без задержек.

2. SO\_KEEPALIVE: включает механизм "поддержания связи". Если соединение не активно в течение определенного времени, система будет отправлять пакеты для проверки, активно ли соединение.

3. SO\_BROADCAST: разрешает отправку широковещательных сообщений. Это полезно для приложений, которые должны отправлять данные всем устройствам в сети.

4. SO\_RCVBUF и SO\_SNDBUF: позволяют установить размер буфера для приема и отправки данных соответственно. Увеличение буфера может помочь при передаче больших объемов данных.

5. TCP\_NODELAY: отключает алгоритм Нагл (Nagle's algorithm), который объединяет небольшие пакеты в один для более эффективной передачи. Это может быть полезно для приложений, требующих низкой задержки.

### Программа для сервера

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

#define SK SO\_KEEPALIVE

#define SL SO\_LINGER

#define TC TCP\_CORK

#define SR SO\_RCVBUF

#define TM TCP\_MAXSEG

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt))) {

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("listen");

exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d\n", PORT);

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

int bytes\_read;

while ((bytes\_read = read(new\_socket, buffer, BUFFER\_SIZE)) > 0) {

// Обработка полученных данных

}

close(new\_socket);

close(server\_fd);

return 0;

}

Листинг 1 - программы ТСР сервера

### Программа для клиента

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

#define SK\_KEEPALIVE SO\_KEEPALIVE

#define SK\_LINGER SO\_LINGER

#define TCP\_CORK TCP\_CORK

#define SO\_RCVBUF SO\_RCVBUF

#define TCP\_MAXSEG TCP\_MAXSEG

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int opt = 1;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

if (setsockopt(server\_fd, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt))) {

perror("setsockopt");

exit(EXIT\_FAILURE);

}

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("listen");

exit(EXIT\_FAILURE);

}

printf("Server is listening on port %d\n", PORT);

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("accept");

exit(EXIT\_FAILURE);

}

int bytes\_read;

while ((bytes\_read = read(new\_socket, buffer, BUFFER\_SIZE)) > 0) {

// Обработка полученных данных

}

close(new\_socket);

close(server\_fd);

return 0;

}

Листинг 2 - программа ТСР клиента

### Тестирование

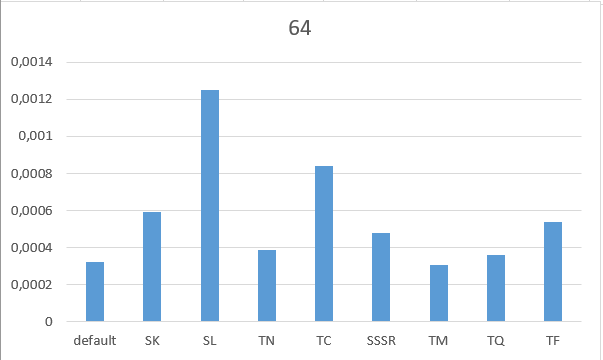


Рисунок 1 – время для 64 б

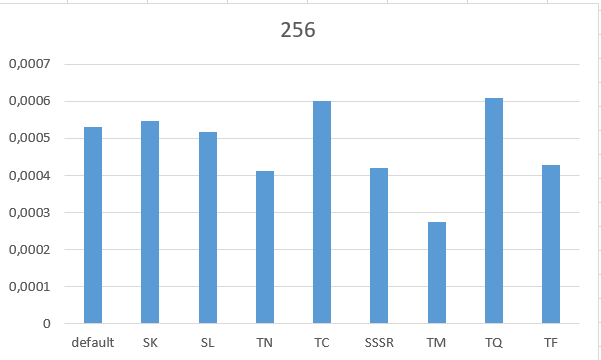


Рисунок 2 – время для 256 б

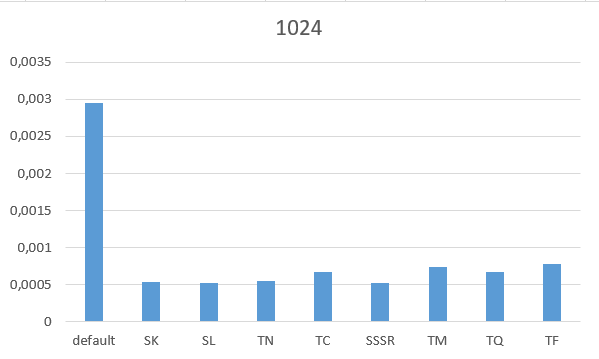


Рисунок 3 – время для 1024 б

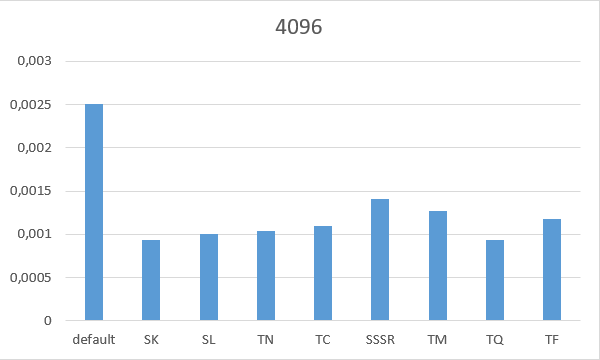


Рисунок 4 – время для 4096 б

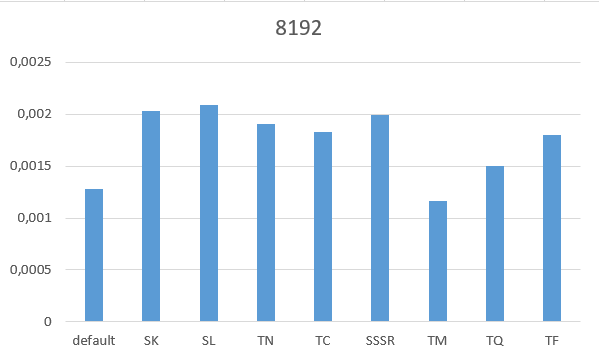


Рисунок 5 – время для 8192 б

# rpc-программа

## Задание

Реализовать rpc-программу для linux с поддержкой аутентификации.

## Ход работы

### Теория

PC (Remote Procedure Call, Сервис вызова удаленных процедур) представляет собой интерфейс между удаленными пользователями и определенными программами хоста, которые запускаются по запросам этих пользователей.

Сервис RPC какого-либо хоста, как правило, предоставляет клиентам комплекс программ. Каждая из таких программ состоит, в свою очередь, из одной или нескольких удаленных процедур.

### Реализация программы

Создадим файл под названием IDL.x. Этот файл нужен для того, чтобы для описать интерфейс взаимодействия между клиентом и сервером. В файле запишем следующий код:

struct auth\_request {

char username[50];

char password[50];

};

struct auth\_response {

int success;

char message[100];

};

program AUTH\_PROG {

version AUTH\_VERS {

auth\_response AUTHENTICATE(auth\_request) = 1;

} = 1;

} = 0x23451111;

Листинг 3 - IDL file

Далее в терминале используем команду rpcgen -a -C IDL.x. Она генерирует дополнительные вспомогательные файлы для полноценной программы RPC (шаблоны клиентского и серверного кода)

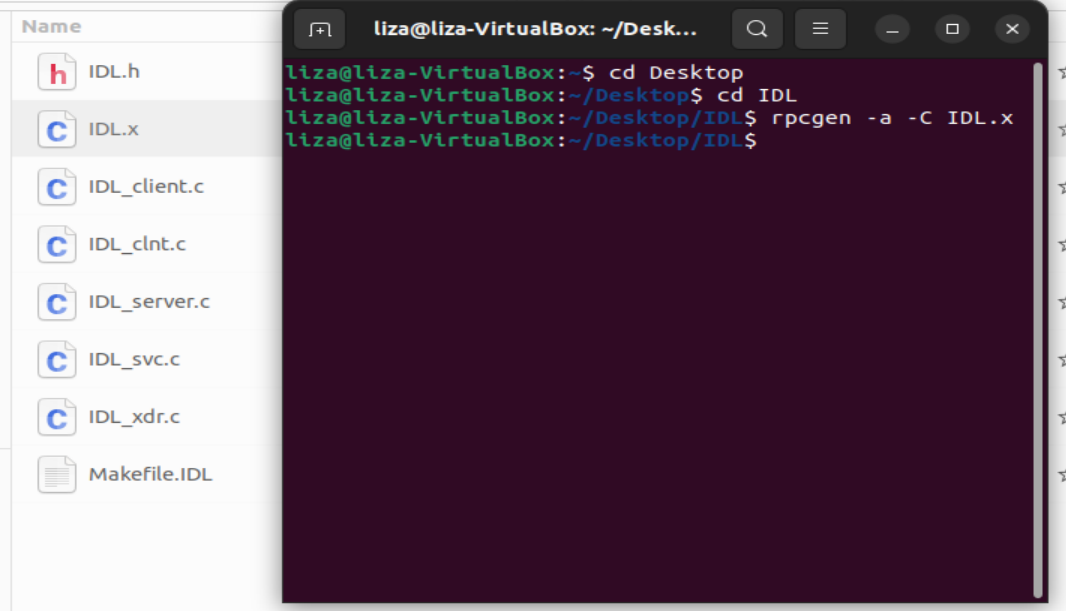


Рисунок 6 – создание вспомогательных файлов

Нужно отредактировать файл IDL\_client.c так, чтобы он подходил под условие задачи, то есть аутентификацию.

#include "IDL.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int main(int argc, char \*argv[]) {

CLIENT \*clnt;

char \*server;

auth\_request req;

auth\_response \*res;

if (argc < 2) {

printf("Usage: %s <server\_host>\n", argv[0]);

exit(1);

}

server = argv[1];

printf("Enter username: ");

scanf("%s", req.username);

printf("Enter password: ");

scanf("%s", req.password);

// Создание клиента

clnt = clnt\_create(server, AUTH\_PROG, AUTH\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror(server);

exit(1);

}

// Вызов удалённой процедуры

res = authenticate\_1(&req, clnt);

if (res == NULL) {

clnt\_perror(clnt, server);

clnt\_destroy(clnt);

exit(1);

}

// Вывод результата

if (res->success) {

printf("Server response: %s\n", res->message);

} else {

printf("Server response: %s\n", res->message);

}

clnt\_destroy(clnt);

return 0;

}

Листинг 4 – код для клиента

Также нужно отредактировать и файл IDL\_server.c.

#include "IDL.h"

#include <stdio.h>

#include <string.h>

int check\_credentials(const char \*username, const char \*password) {

return (strcmp(username, "admin") == 0 && strcmp(password, "password") == 0);

}

auth\_response \*authenticate\_1\_svc(auth\_request \*req, struct svc\_req \*rqstp) {

static auth\_response result;

// Проверка имени пользователя и пароля

if (check\_credentials(req->username, req->password)) {

result.success = 1;

strncpy(result.message, "Authenticated", sizeof(result.message));

} else {

result.success = 0;

strncpy(result.message, "Invalid credentials", sizeof(result.message));

}

return &result;

}

Листинг 5 – код для сервера

Теперь нужно скомпилировать все файлы вместе.

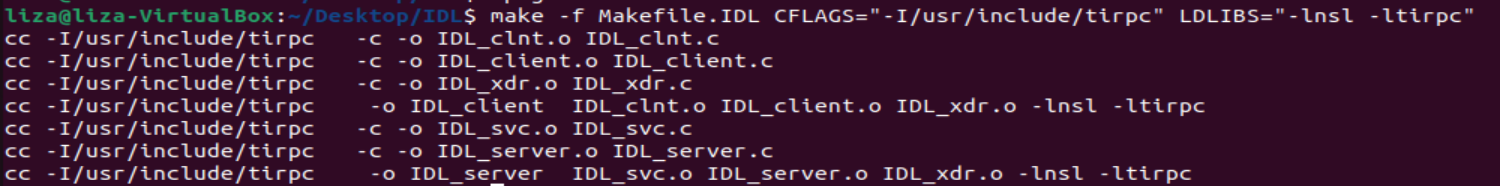


Рисунок 7 – компиляция файлов

Запускаем сервер и клиента, происходит аутентификация.

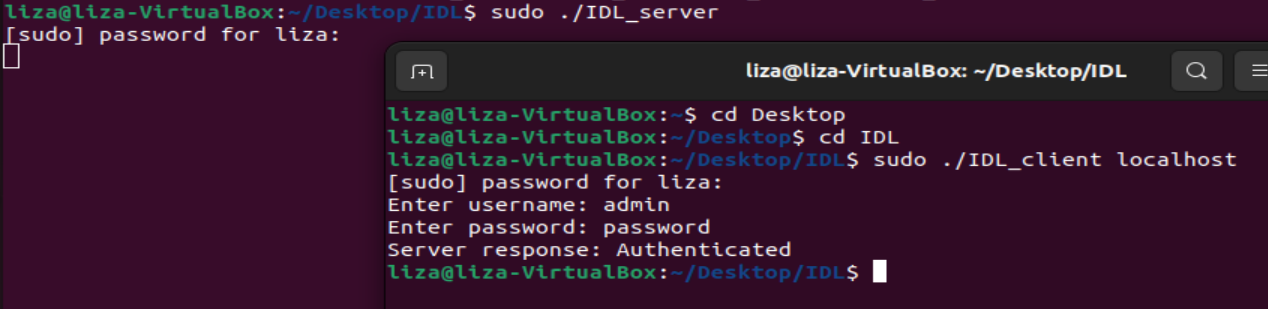


Рисунок 8 - аутентификация

Заключение

В ходе данной лабораторной работы мы познакомились с сетевыми протоколами. Посмотрели как работают сокеты tcp при различных настройках setsockopt и реализовали rpc-программу для linux с поддержкой аутентификации Это позволило закрепить полученные знания и навыки. В ходе выполнения лабораторной работы были выполнены все задачи и достигнуты поставленные цели.

Список использованных источников

1. Эндрю Таненбаум Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с.