**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»**

**Факультет безопасности информационных технологий**

**Дисциплина:**

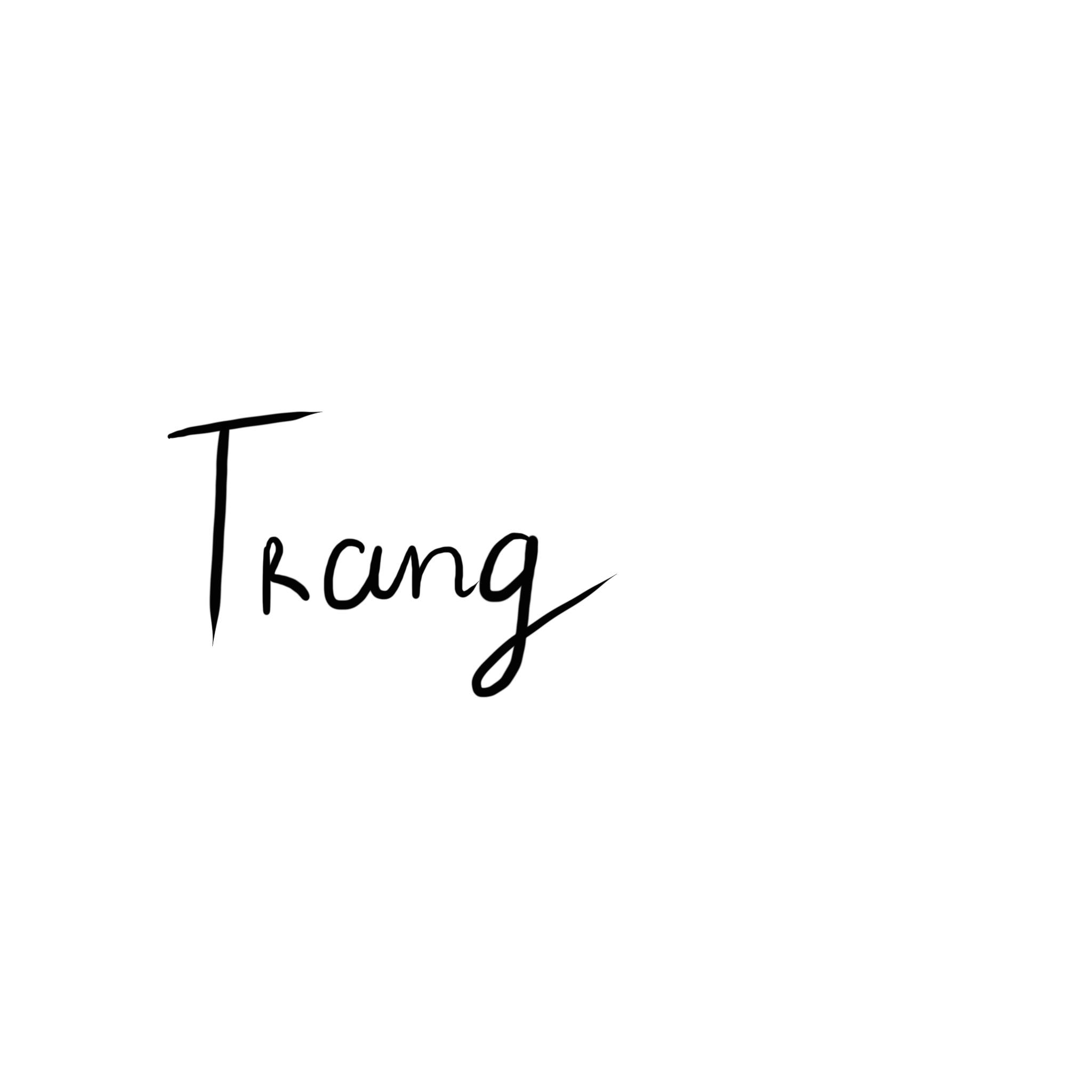
«Операционные системы»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9**

# **Выполнил:**

Ву Тхи Тунг Транг

Группа:N3249





(подпись)

Нгуен Тхи Маи

Группа:N3245



(подпись)

# **Проверил:**

Савков Сергей Витальевич



(подпись)

**Запросы**

1. Простой вариант (одно из):

• Написать фильтр сетевых пакетов на основе nfqueue и iptables и протестировать

скорость работы

• Протестировать работу сокетов tcp при различных настройках setsockopt

2. Сложный вариант (одно из):

• Написать фильтр пакетов на основе интерфейса netfilter

• Реализовать rpc-программу для linux с поддержкой аутентификации

(rpcinfo,rpcbind)

**Ход работы**

**I) Простой вариант: Протестировать работу сокетов tcp при различных настройках**

**setsockopt**

* **server.c**

**#include <stdio.h>**

**#include <string.h> // strlen**

**#include <stdlib.h>**

**#include <sys/socket.h>**

**#include <arpa/inet.h> // inet\_addr**

**#include <unistd.h> // write**

**// The thread function**

**void connection\_handler(void);**

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

**int socket\_desc, client\_sock, c, read\_size;**

**char client\_msg[2000];**

**struct sockaddr\_in server, client;**

**// Create socket**

**socket\_desc = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (socket\_desc == -1) {**

**printf("Could not create socket\n");**

**return 1;**

**}**

**puts("Socket created");**

**// Socket options (uncomment if needed)**

**int optval = 1;**

**//setsockopt(socket\_desc, SOL\_SOCKET, SO\_DEBUG, &optval, sizeof(optval));**

**if (setsockopt(socket\_desc, SOL\_SOCKET, SO\_REUSEADDR, &optval, sizeof(optval)) < 0) {**

**perror("setsockopt(SO\_REUSEADDR) failed");**

**return 1;**

**}**

**// Prepare the sockaddr\_in structure**

**server.sin\_family = AF\_INET;**

**server.sin\_addr.s\_addr = INADDR\_ANY;**

**server.sin\_port = htons(8080);**

**// Bind**

**if (bind(socket\_desc, (struct sockaddr \*)&server, sizeof(server)) < 0) {**

**// Print the error message**

**perror("Bind failed. Error");**

**return 1;**

**}**

**puts("Bind done");**

**// Listen**

**listen(socket\_desc, 3);**

**// Accept an incoming connection**

**puts("Waiting for incoming connections...");**

**c = sizeof(struct sockaddr\_in);**

**client\_sock = accept(socket\_desc, (struct sockaddr \*)&client, (socklen\_t\*)&c);**

**if (client\_sock < 0) {**

**perror("Accept failed");**

**return 1;**

**}**

**puts("Connection accepted");**

**// Receive a message from client**

**while ((read\_size = recv(client\_sock, client\_msg, 2000, 0)) > 0) {**

**// Send the message back to client**

**write(client\_sock, client\_msg, strlen(client\_msg));**

**}**

**if (read\_size == 0) {**

**puts("Client disconnected");**

**fflush(stdout);**

**} else if (read\_size == -1) {**

**perror("Receive failed");**

**}**

**close(socket\_desc);**

**return 0;**

**}**

* **client.c**

**#include <stdio.h> // printf**

**#include <string.h> // strlen**

**#include <sys/socket.h> // socket**

**#include <arpa/inet.h> // inet\_addr**

**#include <unistd.h> // close**

**#include <time.h> // clock**

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

**int sock;**

**struct sockaddr\_in server;**

**char msg[1000], server\_reply[2000];**

**// Create socket**

**sock = socket(AF\_INET, SOCK\_STREAM, 0);**

**if (sock == -1) {**

**printf("Could not create socket\n");**

**return 1;**

**}**

**puts("Socket created");**

**server.sin\_addr.s\_addr = inet\_addr("127.0.0.1");**

**server.sin\_family = AF\_INET;**

**server.sin\_port = htons(8080);**

**// Connect to remote server**

**if (connect(sock, (struct sockaddr \*)&server, sizeof(server)) < 0) {**

**perror("Connect failed. Error");**

**return 1;**

**}**

**puts("Connected\n");**

**// Keep communicating with server**

**printf("Enter command: ");**

**scanf("%s", msg);**

**clock\_t start = clock();**

**for (int k = 0; k < 100; k++) {**

**send(sock, msg, strlen(msg), 0);**

**}**

**recv(sock, server\_reply, 2000, 0);**

**clock\_t stop = clock();**

**// Calculate and display the time taken for the test**

**float time = ((float)(stop - start) \* 1000) / CLOCKS\_PER\_SEC;**

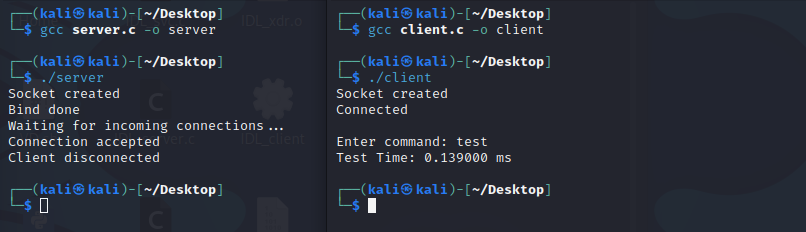
**printf("Test Time: %lf ms\n", time);**

**close(sock);**

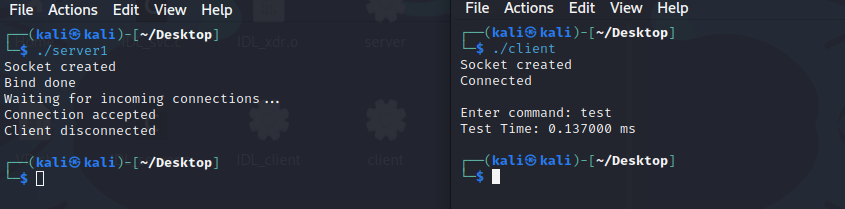
**return 0;**

**}**

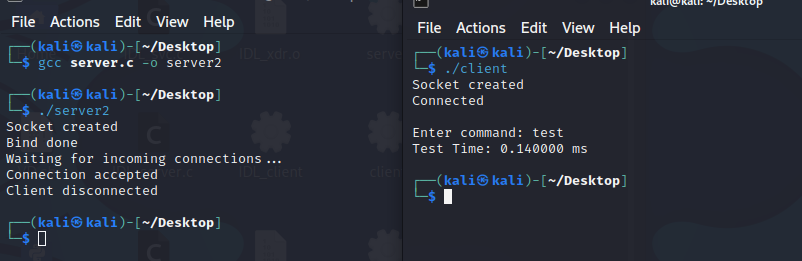
* Для изменения настроек была использована команда setsockopt() в Server\_code.c
* Изменим параметр SO\_REUSEADDR, который как понятно из названия позволяет использовать адреса повторно. А также параметр SO\_DEBUG, отвечающий за запись отладочной информации.
* Тест со стандартными настройками

****

* Тест с параметром SO\_REUSEADDR

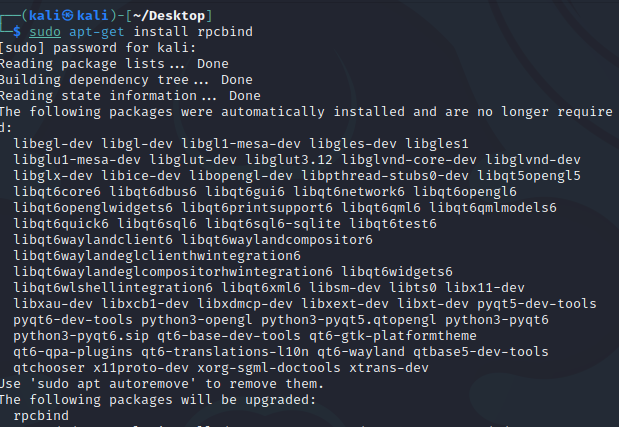


* Тест с параметром SO\_DEBUG

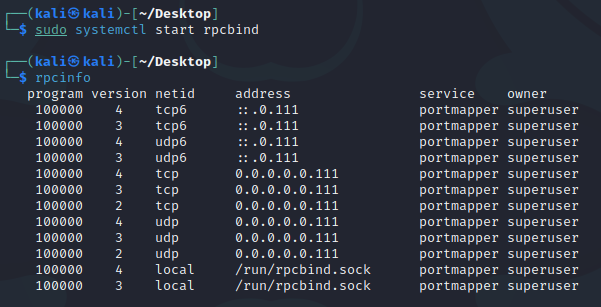


**II) Сложный вариант: Реализовать rpc-программу для linux с поддержкой аутентификации (rpcinfo, rpcbind)**

* Шаг 1: Установка rpcbind и проверка rpcinfo

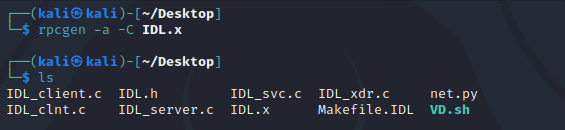


* rpc



* Шаг 2: Написание и компиляция файла IDL
* /\*Structure to hold the 2 values to be used in computation\*/
* struct values{
* float num1;
* float num2;
* char operation;
* };
* /\*Programm, version and procedure definition\*/
* program COMPUTE { //name of the programm.
* version COMPUTE\_VERS{ //the name of the programm version.
* float ADD(values) =1;
* float SUB(values)=2;float MUL(values)=3;
* float DIV(values)=4;
* } =6; // Arbitrary number used for identification
* } = 456123789; // Arbitrary number used for identification
* Шаг 3: Скомпилированные с помощью компилятора rpcgen

После этого, я получил несколько файлов:



* Шаг 4: Редактирование клиентских и серверных программ

**IDL\_client.c**

#include "IDL.h"

#include <stdio.h>

float compute\_6(char \*host,float a,float b,char op){

CLIENT \*clnt;

float \*result\_1;

values add\_6\_arg;

float \*result\_2;

values sub\_6\_arg;

float \*result\_3;

values mul\_6\_arg;

float \*result\_4;

values div\_6\_arg;

if(op=='+'){

add\_6\_arg.num1=a;

add\_6\_arg.num2=b;

add\_6\_arg.operation=op;

clnt = clnt\_create (host, COMPUTE, COMPUTE\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

result\_1 = add\_6(&add\_6\_arg, clnt);

if (result\_1 == (float \*) NULL) {

clnt\_perror (clnt, "call failed");

}

clnt\_destroy (clnt);

return (\*result\_1);

}

else if(op=='-'){

sub\_6\_arg.num1=a;

sub\_6\_arg.num2=b;

sub\_6\_arg.operation=op;

clnt = clnt\_create (host, COMPUTE, COMPUTE\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

result\_2 = sub\_6(&sub\_6\_arg, clnt);

if (result\_2 == (float \*) NULL) {

clnt\_perror (clnt, "call failed");

}

clnt\_destroy (clnt);

return (\*result\_2);

}

else if(op=='\*'){

mul\_6\_arg.num1=a;

mul\_6\_arg.num2=b;

mul\_6\_arg.operation=op;

clnt = clnt\_create (host, COMPUTE, COMPUTE\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

result\_3 = mul\_6(&mul\_6\_arg, clnt);

if (result\_3 == (float \*) NULL) {

clnt\_perror (clnt, "call failed");

}

clnt\_destroy (clnt);

return (\*result\_3);

}

else if(op=='/'){

if(b==0){

printf("You are trying to divide by zero. Please insert a valid number.\n");

exit(0);

}

else{

div\_6\_arg.num1=a;

div\_6\_arg.num2=b;

div\_6\_arg.operation=op;

clnt = clnt\_create (host, COMPUTE, COMPUTE\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

result\_4 = div\_6(&div\_6\_arg, clnt);

if (result\_4 == (float \*) NULL) {

clnt\_perror (clnt, "call failed");

}

clnt\_destroy (clnt);

return (\*result\_4);

}

}

}

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

char \*host;

float number1,number2;

char oper;

printf("Enter the 2 numbers followed by the operation to perform:\n");

scanf("%f",&number1);

scanf("%f",&number2);

scanf("%s",&oper);

host = argv[1];

printf("Answer= %f\n",compute\_6 (host,number1,number2,oper));

exit(0);

}

**IDL\_server.c**

#include "IDL.h"

#include <stdio.h>

float \*add\_6\_svc(values \*argp, struct svc\_req \*rqstp){

static float result;

result = argp->num1 + argp->num2;

return &result;

}

float \*sub\_6\_svc(values \*argp, struct svc\_req \*rqstp){

static float result;

result = argp->num1 - argp->num2;

return &result;

}

float \*mul\_6\_svc(values \*argp, struct svc\_req \*rqstp){

static float result;

result = argp->num1 \* argp->num2;

return &result;

}

float \*div\_6\_svc(values \*argp, struct svc\_req \*rqstp){

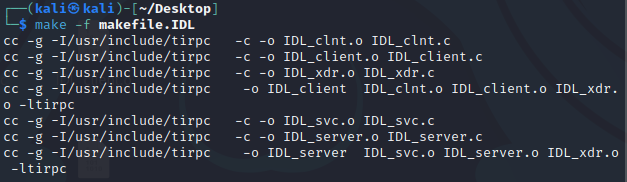
static float result;

result = argp->num1 / argp->num2;

return &result;

}

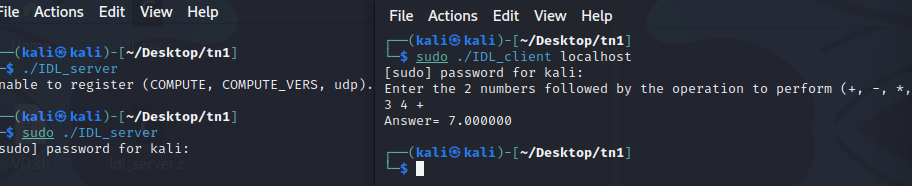
* Шаг 5: Компиляция всех файлов



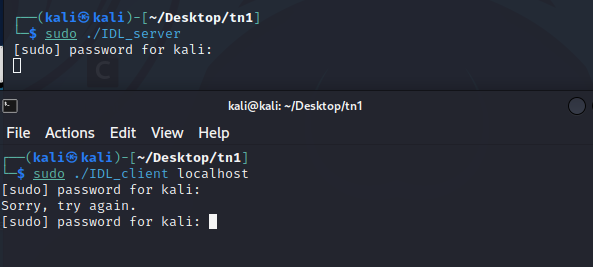
* Шаг 6: Запуск сервера и клиента

Я открыл другой терминал и запустил клиент там.

Авторизация прошла успешно, и программа дала верный ответ



Попробуем зайти под другим логином и ввести неверный пароль



**II) Вывод:**

* В ходе работы были проведены тесты работы сокетов TCP при различных настройках setsockopt. Как уже было сказано, для данных случаев диапазон значений сильно разница, и ощутимого влияния на производительность

ничем не оказывалось, однако в реальных ситуациях воздействие должно

быть более ощутимо.

* Firewalling решает, каким пакетам разрешено входить и выходить из системы. firewalling обеспечивается с помощью netfilter. iptables - это просто интерфейс для netfilter.
* Если INPUT цепочка не означает более безопасную, потому что при работе сети используются как INPUT, так и OUTPUT.
* После выполнения работы мы написали rpc-программу на linux с поддержкой аутентификации.