|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_

**Лабораторная работа № 7**

**По дисциплине: “Операционные системы”**

|  |  |
| --- | --- |
| **Тема:** “Сокеты в пространстве файловых имен и сетевые сокеты”  **Студент:** Мередова Айджахан  **Группа:** ИУ7-66Б  **Преподаватель:** Рязанова Н.Ю. |  |

Москва.

2021 г.

**Задание на лабораторную работу**

Лабораторная работа состоит из двух частей:

1. Организовать взаимодействие параллельных процессов на отдельном компьютере.
2. Организовать взаимодействие параллельных процессов в сети (ситуацию моделируем на одной машине).

**Задание1.**

Написать приложение по модели клиент-сервер, демонстрирующее взаимодействие параллельных процессов на отдельном компьютере с использованием сокетов в файловом пространстве имен: семейство - AF\_UNIX, тип - SOCK\_DGRAM. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

**Задание2.**

Написать приложение по модели клиент-сервер, осуществляющее взаимодействие параллельных процессов, которые выполняются на разных компьютерах. Для взаимодействия с клиентами сервер должен использовать мультиплексирование. Сервер должен обслуживать запросы параллельно запущенных клиентов. При демонстрации работы программного комплекса необходимо запустить несколько клиентов (не меньше 5) и продемонстрировать, что сервер обрабатывает обращения каждого запущенного клиента.

**Задание1.**

**Листинг кода программы:**

**client.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <unistd.h>

#define SOCKET\_NAME "sock.soc"

#define BUF\_LEN 256

int main(int argc, char \*\*argv)

{

struct sockaddr\_un server\_addr; //адрес сервера

int c\_pid; //pid клиента

char message[BUF\_LEN];

// Создание сокета в файловом пространстве имен

// Домен -- обозн. тип соединения AF\_UNIX (локальн. соед-е)

// Тип сокета -- SOCK\_DGRAM -датаграммный сокет

// Протокол -- 0 - протокол выбирается по умолчанию

int sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (sock < 0)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

//Семейство адресов, которыми мы будем пользоваться

server\_addr.sun\_family = AF\_UNIX;

//Имя файла сокета

strcpy(server\_addr.sun\_path, SOCKET\_NAME);

// Получаем pid клиента

c\_pid = getpid();

while (1)

{

printf("== Сообщение клиента (pid = %d) == : ", c\_pid);

fgets(message, BUF\_LEN, stdin);

printf("\n");

// отправка серверу

// Первый пар-р – дескриптор сокета,

// второй и третий пар-ры - адрес буфера для передачи данных и его длина.

// Четвертый пар-р - для передачи дополнительных флагов.

// Предпоследний и последний пар-ры - адрес сервера и его длина

sendto(sock, message, strlen(message), 0,

(struct sockaddr \*)&server\_addr, sizeof(server\_addr));

}

return EXIT\_SUCCESS;

}

**server.c**

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <sys/socket.h>

#include <sys/un.h>

#include <signal.h>

#define SOCKET\_NAME "sock.soc"

#define BUF\_LEN 256

int sock;

void catch\_ctrlc(int signum)

{

printf("Получен сигнал Ctrl+C - остановка работы сервера\n");

close(sock);

//Перед выходом из программы-сервера следует удалить файл сокета, созданный в результате вызова socket()

unlink(SOCKET\_NAME);

}

int main()

{

// Для сокетов Unix (сокетов в файловом пространстве имен)

// есть специализированная структура sockaddr\_un - адрес сервера

struct sockaddr\_un server\_addr;

char message[BUF\_LEN];

int er\_flag = 0;

int message\_len = 0;

// Создание сокета в файловом пространстве имен

// Домен -- обозн. тип соединения AF\_UNIX (локальн. соед-е)

// адреса интерпретируются как имена файлов в UNIX.

// Тип сокета -- SOCK\_DGRAM означает датаграммный сокет сохраняющий границы сообщений

// В пространстве файловых имен датаграммные сокеты также надежны, как и потоковые сокеты.

// Протокол -- 0, протокол выбирается по умолчанию

// (домен, тип, протокол), возвр дескриптор сокета

sock = socket(AF\_UNIX, SOCK\_DGRAM, 0);

if (sock < 0)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

server\_addr.sun\_family = AF\_UNIX; // семейство адресов, которыми мы будем пользоваться

strcpy(server\_addr.sun\_path, SOCKET\_NAME); // имя файла сокета

// bind(дескриптор сокета, указатель на структуру, длина структуры)

// связывает дескриптор сокета к заданному адресу. (присв-е имени)

// struct sockaddr

// {

// unsigned short sa\_family; // Семейство адресов, AF\_xxx

// char sa\_data[14]; // 14 байтов для хранения адреса

if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

// отслеживаем сигнал

signal(SIGINT, catch\_ctrlc);

// программа-сервер доступна для соединения по заданному адресу (имени файла)

while (!er\_flag)

{

// Для чтения данных из датаграммного сокета - recvfrom,

// блокирует программу,пока на входе не появятся новые данные

// Так как нас не интересуют данные об адресе клиента - NULL в предпоследнем и последнем параметрах

if ((message\_len = recvfrom(sock, message, BUF\_LEN, 0, NULL, NULL)) < 0)

{

printf("%s", strerror(errno));

er\_flag = 1;

}

message[message\_len] = 0;

printf("== Получено сообщение == : %s\n", message);

}

// Закрываем сокет

close(sock);

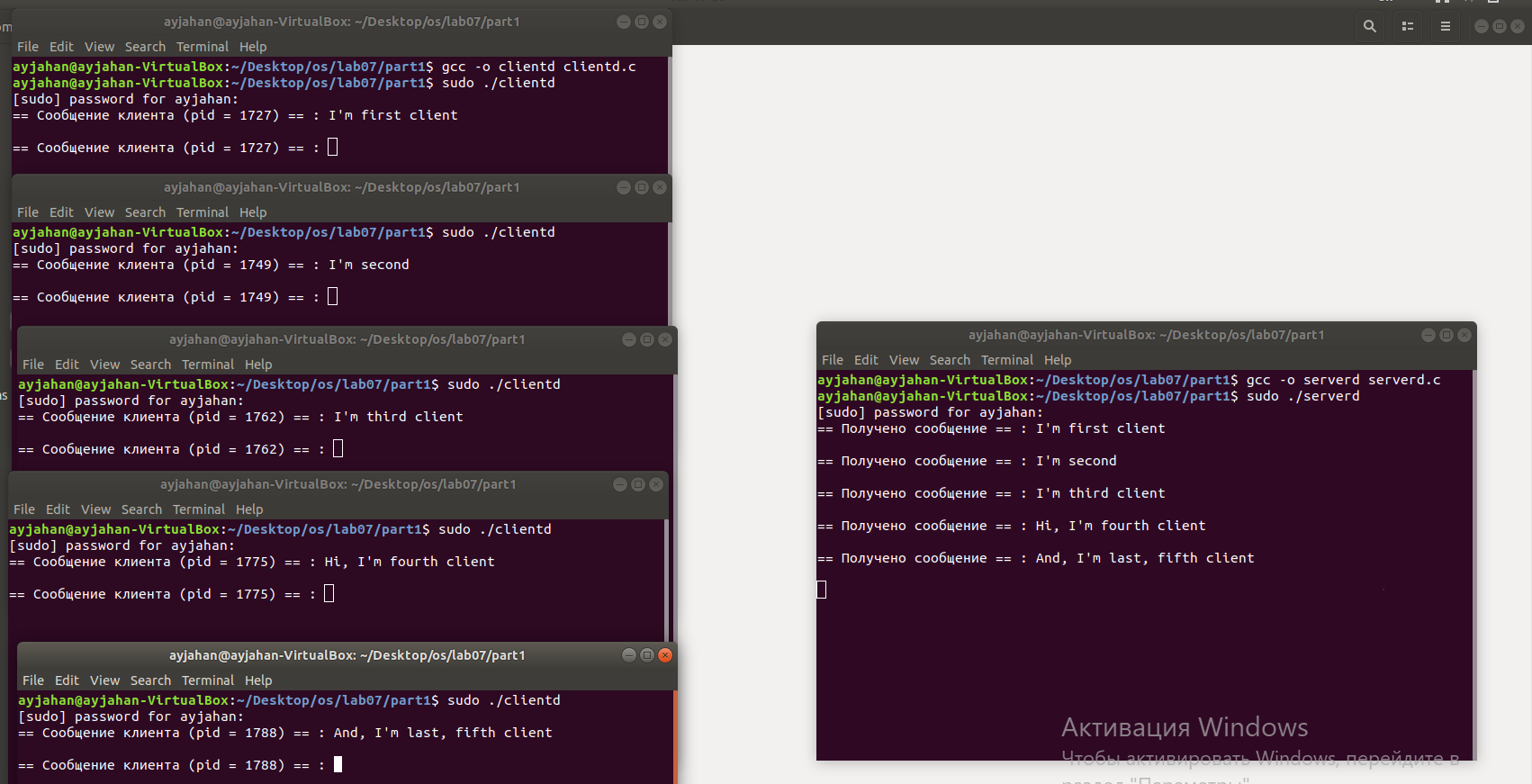
// Удаляем файл сокета

unlink(SOCKET\_NAME);

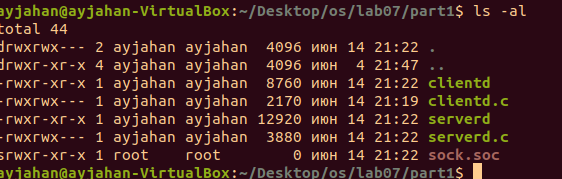
return -er\_flag;

}

**Результат работы программы:**



Командой ls –al может посмотреть созданный файл sock.soc



**Задание 2.**

**Листинг кода программы:**

**client.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <unistd.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <errno.h>

#include <sys/select.h>

#define MSG\_LEN 256

int sock;

int stop\_flag = 0;

void sigint\_catcher(int signum)

{

stop\_flag = 1;

}

int main(int argc, char \*\*argv)

{

//создать сокет, не привязан к адресу

//AF\_INET - сокет должен быть сетевым

//SOCK\_STREAM сокет потоковый

//Обеспечивает создание двусторонних, надёжных потоков байтов на основе

//установления соединения. Может поддерживаться механизм внепоточных данных.

//не сохраняют границы записей. Потоковый сокет должен быть в состоянии соединения

//перед тем, как из него можно будет отсылать данные или принимать их.

// 0 - протокол по умолчанию

// (домен, тип, протокол), возвр дескриптор сокета

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

if (sock < 0)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

// преобразование доменного имени сервера в его сетевой адрес

struct hostent \*host = gethostbyname("localhost");

// Функция получает указатель на строку с Интернет-именем сервера и возвращает указатель

// на структуру hostent, которая содержит имя сервера в

// приемлемом для дальнейшего использования виде. При этом, если необходимо,

// выполняется разрешение доменного имени в сетевой адрес.

if (!host)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

// инициализировать адрес сокета сервера

struct sockaddr\_in server\_addr;

server\_addr.sin\_family = AF\_INET; // семейство адресов

// Укажем значение порта. Функция htons() переписывает

// двухбайтовое значение порта так, чтобы порядок байтов

// соответствовал принятому в Интернете

// берем номер порта , который больше , чем 1023 - нам не нужен зарезервированный порт.

// Он также больше 5000 и меньше 49152, чтобы не было конфликта с динамическими портами.

server\_addr.sin\_port = htons(42317);

// Укажем адрес (наша программа-сервер зарегистрируется на всех адресах

// машины, на которой она выполняется)

server\_addr.sin\_addr = \*((struct in\_addr \*)host->h\_addr\_list[0]);

// установка соединения с сервером

// если сервер сокет не был связан с адресом, коннект автоматически вызовет сис ф-цию bind

//неявное связывание

//int connect(int s, char \* name, int namelen);

//Первый арг-т - сокет-дескриптор клиента.

//Второй арг-т - указатель на адрес сервера (структура sockaddr) для соответств. домена.

//Третий арг-т - целое число - длина структуры адреса.

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

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

char message[MSG\_LEN];

int message\_num = 0;

srand(time(NULL));

while (!stop\_flag)

{

memset(message, 0, MSG\_LEN);

//Функция inet\_ntoa() преобразует IP-адрес in, заданный в сетевом порядке расположения байтов, в стандартный строчный вид, из номеров и точек.

// Строка располагается в статически размещенном буфере; последующие вызовы перепишут его данные.

// ntohs - переводит данные из узлового порядка расположения байтов в сетевой

sprintf(message, "%d %s отправил серверу %s:%d", message\_num, "Client message", inet\_ntoa(server\_addr.sin\_addr), ntohs(server\_addr.sin\_port));

printf("%s\n", message);

// передача данных

//(дескриптор сокета, адрес буфеа дял передачи данных и его длина)

if (send(sock, message, strlen(message), 0) < 0)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

message\_num++;

sleep(4 + rand() % 6);

}

return EXIT\_SUCCESS;

}

**server.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <sys/select.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <errno.h>

#include <sys/select.h>

#define BUF\_LEN 256

#define SERVER\_SOCK\_PORT 42317

#define MAX\_COUNT 8

int sock;

int sock\_flag = 0;

// прочитать сообщение от клиента

void read\_message(unsigned int client\_id, int \*client\_sockets)

{

char message[BUF\_LEN];

memset(message, 0, BUF\_LEN);

struct sockaddr\_in client\_addr;

int client\_addr\_size = sizeof(client\_addr);

//Getpeername возвращает имя подключившегося сокета. Последний пар-р namelen - объем памяти, который занимает name.

//По возвращении он содержит размер памяти, занимаемый именем машины (байт). Имя не считывается, если буфер слишком мал.

//int getpeername(int s, struct sockaddr \*name, socklen\_t \*namelen);

getpeername(client\_sockets[client\_id], (struct sockaddr \*)&client\_addr, (socklen\_t \*)&client\_addr\_size);

// читаем сообщение (при неудаче считаем что клиент отключился)

int rec\_BUF\_LEN = recv(client\_sockets[client\_id], message, BUF\_LEN, 0);

if (rec\_BUF\_LEN == 0)

{

printf("Клиент отключился: адрес = %s:%d!\n", inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

client\_sockets[client\_id] = 0;

close(client\_sockets[client\_id]);

}

else

{

message[rec\_BUF\_LEN] = '\0';

printf("Сообщение: %s от клиента %s:%d\n",message, inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

}

}

int main(void)

{

struct sockaddr\_in server\_addr;

//создать сокет,

//AF\_INET - сокет должен быть сетевым

//SOCK\_STREAM сокет потоковый

//Обеспечивает создание двусторонних, надёжных потоков байтов на основе

//установления соединения. Может поддерживаться механизм внепоточных данных.

//не сохраняют границы записей. Потоковый сокет должен быть в состоянии соединения

//перед тем, как из него можно будет отсылать данные или принимать их.

// 0 - протокол по умолчанию

// (домен, тип, протокол), возвр дескриптор сокета

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

if (sock < 0)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

// инициализировать адрес сокета-сервера

// struct sockaddr\_in

// {

// short int sin\_family; // Семейство адресов

// unsigned short int sin\_port; // Номер порта

// struct in\_addr sin\_addr; // IP-адрес

// unsigned char sin\_zero[8]; // Дополнение до размера структуры sockaddr

// };

// семейство адресов, которыми мы будем пользоваться

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

//преобразовать число из порядка хоста в сетевой

server\_addr.sin\_port = htons(SERVER\_SOCK\_PORT);

//программа сервер зарегистрируется на всех адресах той машины,

//на которой она выполняется.

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

// привязать сокет приложения-сервера к адресу

if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0)

{

printf("%s", strerror(errno));

return EXIT\_FAILURE;

}

//listen(2) - переводит сервер в режим ожидания запроса на соединение

if (listen(sock, MAX\_COUNT) < 0)

{

printf("%s", strerror(errno));

close(sock);

return EXIT\_FAILURE;

}

int client\_sockets[MAX\_COUNT] = {0};

while (!sock\_flag)

{

// пулл дескрипторов сокетов

fd\_set rfds;

//очищает набор

FD\_ZERO(&rfds);

//добавляет заданный дескриптор из набора.

FD\_SET(sock, &rfds);

int max\_fd = sock;

for (int i = 0; i < MAX\_COUNT; i++)

{

int fd = client\_sockets[i];

if (fd > 0)

FD\_SET(fd, &rfds);

max\_fd = (fd > max\_fd) ? (fd) : (max\_fd);

}

// ждем нового соединения или нового сообщения

// int pselect(int n, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, const struct timespec \*timeout, sigset\_t \* sigmask);

// n на единицу больше самого большого номера файловый дескриптор из всех наборов.

// Первый параметр функции – количество проверяемых дескрипторов.

// Второй, третий и четвертый п-ры - наборы дескр-ов, к. следует проверять на готовность к чтению, записи и на наличие исключительных ситуаций.

// ждать вечно - timeval = NULL

// процесс блокируется в ожидании обновления, выходим из селекта когда в каком-либо из файлов соотв дескр появились новые символы

int updates\_cl\_count = pselect(max\_fd + 1, &rfds, NULL, NULL, NULL, NULL);

//после выхода из pselect в rfds остались только файлы с обновлениями

// если новое соединение есть

// если сокет остался в наборе, то к нему пришло новое подключение

if (FD\_ISSET(sock, &rfds))

{

// принять соединение с клиентом

struct sockaddr\_in client\_addr;

int client\_addr\_size = sizeof(client\_addr);

//используется для получения нового сокета для нового входящего соединения.

//которая устанавливает соединение в ответ на запрос клиента и создает копию сокета для того,

//чтобы исходный сокет мог продолжать прослушивание

int accepted\_sock = accept(sock, (struct sockaddr \*)&client\_addr, (socklen\_t \*)&client\_addr\_size);

// Второй параметр функции accept() содержит сведения об адресе клиента, запросившего соединение,

// а третий параметр указывает размер второго.

// не интересуют данные об адресе клиента -> передаем значения NULL в предпоследнем и последнем параметрах

//неявное связывание с адресом

if (accepted\_sock < 0)

{

printf("%s", strerror(errno));

close(sock);

return EXIT\_FAILURE;

}

// обновляем информацию о наборе клиентов

int added\_flag = 0;

for (int i = 0; i < MAX\_COUNT && !added\_flag; i++)

{

// находим свободное место в таблице клиентов

if (client\_sockets[i] == 0)

{

client\_sockets[i] = accepted\_sock;

added\_flag = 1;

}

}

printf("Новое подключение: fd = %d, address = %s:%d\n", accepted\_sock, inet\_ntoa(client\_addr.sin\_addr), ntohs(client\_addr.sin\_port));

}

// проверяем есть ли полученные сообщения

for (int i = 0; i < MAX\_COUNT; i++)

{

int fd = client\_sockets[i];

if ((fd > 0) && FD\_ISSET(fd, &rfds))

{

read\_message(i, client\_sockets);

}

}

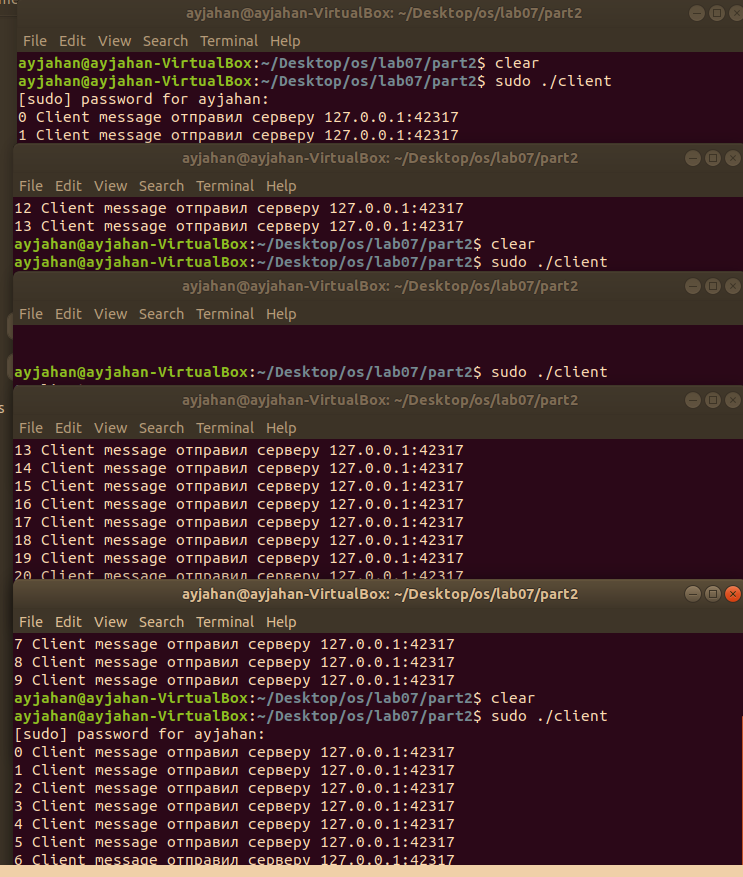
}

return 0;

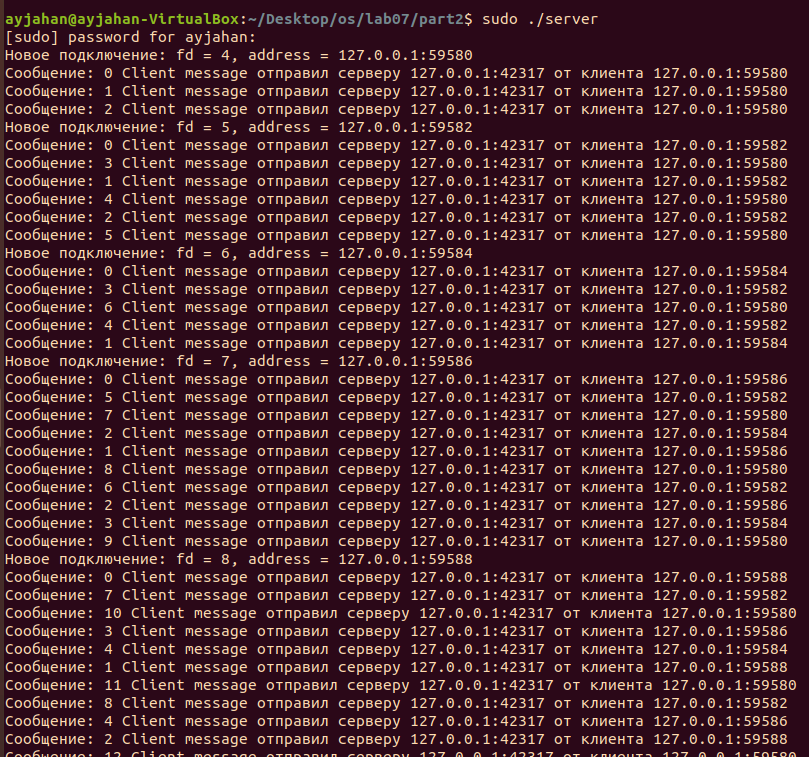
}

**Результат работы программы:**

Клиенты



Сервер



Порты

