1. МИНОБРНАУКИ РОССИИ
2. САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
3. ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
4. «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
5. Кафедра Вычислительной техники

ОТЧЁТ

по лабораторной работе №11

по дисциплине «Организация процессов и программирования в среде Linux»

1. Тема: ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ ЧЕРЕЗ СОКЕТЫ

|  |  |  |
| --- | --- | --- |
| Студент гр. 9308 |  | Соболев М.С. |
| Преподаватель |  | Разумовский Г.В. |

Оглавление

[1. Введение 3](#__RefHeading___Toc336_311249038)

[1.1. Введение 3](#__RefHeading___Toc15338_3795591338)

[1.2. Порядок выполнения работы 3](#__RefHeading___Toc15340_3795591338)

[2. Тексты программ 5](#__RefHeading___Toc3484_1927017605)

[2.1. server.cpp 5](#__RefHeading___Toc3486_1927017605)

[2.2. subserver.cpp 10](#__RefHeading___Toc3488_1927017605)

[2.3. client.cpp 13](#__RefHeading___Toc3490_1927017605)

[3. Скриншоты работы каждой программы 19](#__RefHeading___Toc3492_1927017605)

[4. Вывод 24](#__RefHeading___Toc358_311249038)

[5. Список использованных источников 25](#__RefHeading___Toc360_311249038)

# 1. Введение

## 1.1. Введение

Тема работы: Взаимодействие процессов через сокеты.

Цель работы: Знакомство с механизмом взаимодействия процессов через сокеты.

## 1.2. Порядок выполнения работы

1. Написать две программы, которые обмениваются сообщениями через дейтаграммные сокеты и выводят принятые сообщения на экран. Первая программа посылает второй сообщение (данные, полученные от функции uname), а затем ждёт от неё ответа. Вторая программа принимает сообщение, а затем посылает ответ первой программе (свой IP-адрес). Если обе программы не получают сообщение в течение определённого времени, которое задаётся при запуске обеих программ, то они заканчивают работу с уведомлением о времени ожидания и непринятии сообщения.

2. Написать две программы (сервер и клиент), которые обмениваются сообщениями через потоковые сокеты. Сервер может принимать последовательно сообщения от нескольких клиентов и, если в течение определённого времени после обращения последнего клиента новые запросы на соединения и приём сообщения от клиентов не поступают, то сервер завершает свою работу с выводом на экран времени ожидания, которое задаётся при запуске программы. Клиенты запрашивают у сервера текущие дату и время и сообщает ему своё имя, которое задаётся при запуске клиента. Сервер выводит на экран имя клиента и передаёт ему дату и время. Полученный ответ клиент выводит на экран и завершает свою работу.

3. Написать две программы (сервер и клиент), которые обмениваются сообщениями через потоковые сокеты. Клиенты проверяют возможность соединения с сервером и в случае отсутствия соединения или истечения времени ожидания отправки сообщения завершают работу. После соединения с сервером они генерируют случайную последовательность чисел и выводят её на экран, а затем отсылают серверу. Сервер в течение определённого времени ждёт запросы от клиентов и в случае их отсутствия завершает работу. При поступлении запроса от клиента сервер порождает обслуживающий процесс, который принимает последовательность чисел, упорядочивает её и выводит на экран, а затем отсылает обратно клиенту и завершают работу. Клиент полученную последовательность выводит на экран и заканчивает свою работу.

Выбранные задания: 3.

# 2. Тексты программ

## 2.1. server.cpp

/\*

\* ./server

\*

\* could be only 1 server

\*

\*/

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

using namespace std;

typedef struct sockaddr\_in SocketAddressIn; // struct with server address w/ IP

typedef struct sockaddr SocketAddress; // struct with server address

typedef struct timeval TimeValue; // struct w/ max waiting time to sending a message

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

{

// ---------- INITIALIZING & PREPARING ----------

int socket\_fd; // socket-accepter, who gets the socket, who wants to establish connection w/ server

int socket\_listener; // socket-listener, who listens all requests to server

int socket\_detect; // socket change check

char socket\_client\_name[255];

fd\_set fds; // set of the handles

pid\_t process\_id; // new process id

SocketAddressIn socket\_address\_in;

TimeValue time\_value;

// ---------- SOCKET CREATION ----------

// creating socket-listener, who gets all the requests to server

// setting the connetion port, protocol IPv4 and address

// then binding the settings (port, protocol IPv4 and address) for it,

// so the others can identify it now (before that we haven't exact address for it,

// and the others couldn't know, how to connect)

/\*

https://man7.org/linux/man-pages/man2/socket.2.html

int socket(int domain, int type, int protocol)

creates socket (w/ domain in domain, protocol in protocol), return file descriptor for the new socket if successful, -1 if error

AF\_INET -- IPv4

SOCK\_STREAM -- two endpoints bytes stream

\*/

socket\_listener = socket(AF\_INET, SOCK\_STREAM, 0); // creating recieving socket

if(socket\_listener < 0)

{

perror("Socket creation error");

exit(1);

}

socket\_address\_in.sin\_family = AF\_INET; // network interaction, IPv4 protocol

socket\_address\_in.sin\_port = htons(3434); // port number in network byte order

socket\_address\_in.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // ip-address; "127.0.0.1" is "localhost"

/\*

https://man7.org/linux/man-pages/man2/bind.2.html

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)

bind (associate) a name (address), written in sockaddr, to a socket, return 0 if success, -1 if error

\*/

if(bind(socket\_listener, (SocketAddress\*)&socket\_address\_in, sizeof(socket\_address\_in)) < 0) // binding to a network address

{

perror("Binding to the network address error");

exit(2);

}

// ---------- RECIEVING MESSAGE FROM CLIENTS & SENDING MESSAGE TO CLIENTS ----------

// setting timeout time for waiting any requests from clients,

// every time that we establishing a new connection, we reset it to 0,

// so count 15 secongs again,

// then we establishing max connections -- 10 -- for server

// and finally we waiting any requests from socket-listener, accept it w/ socket-accepter

// and start a new process, who treat a message and send it backwards

// setting waiting time for recieving a message from client

time\_value.tv\_sec = 15;

time\_value.tv\_usec = 0;

/\*

https://man7.org/linux/man-pages/man2/listen.2.html

int listen(int sockfd, int backlog)

listen for any connections to the socket w/ maximal queue for connections requests, return 0 if success, -1 if error

\*/

listen(socket\_listener, 10); // creating a queue for the attached sockets

while(true)

{

// waiting a socket

FD\_ZERO(&fds);

FD\_SET(socket\_listener, &fds);

time\_value.tv\_sec = 15;

time\_value.tv\_usec = 0;

/\*

https://man7.org/linux/man-pages/man2/select.2.html

int select(int nfds, fd\_set \*restrict readfds, fd\_set \*restrict writefds, fd\_set \*restrict exceptfds, struct timeval \*restrict timeout)

void FD\_SET(int fd, fd\_set \*set)

void FD\_ZERO(fd\_set \*set)

monitoring file descriptors to find ready for I/O operations, return number of file descriptors contained in the three returned descriptor sets/0 if timeout if success, -1 if error

\*/

socket\_detect = select(FD\_SETSIZE, &fds, NULL, NULL, &time\_value); // socket state check, number of asked handles, ready to read checking, time

if(socket\_detect == 0)

{

cout << "---------- MESSAGE REQUEST FROM CLIENT TIMEOUT ----------\n";

/\*

https://man7.org/linux/man-pages/man2/close.2.html

int close(int fd)

closing a file descriptor of the socket, return 0 if success, -1 if error

\*/

close(socket\_fd);

exit(1);

}

else

{

process\_id = -1;

if (process\_id != 0)

{

/\*

https://man7.org/linux/man-pages/man2/accept.2.html

int accept(int sockfd, struct sockaddr \*restrict addr, socklen\_t \*restrict addrlen)

accepts a connection to the socket (extracts the 1st connection on a queue)

return a file descriptor for the accepted socket (int >0) if success, -1 if error

\*/

socket\_fd = accept(socket\_listener, NULL, NULL);

if(socket\_fd < 0)

{

perror("Connection establish error");

exit(1);

}

sprintf(socket\_client\_name, "%d", socket\_fd);

process\_id = fork();

if(process\_id == 0)

{

execl("subserver", " ", socket\_client\_name, NULL);

}

}

}

}

// ---------- CLEANING & TERMINATING ----------

if (process\_id != 0) // if it's NOT a new process, close ITS socket

{

/\*

https://man7.org/linux/man-pages/man2/close.2.html

int close(int fd)

closing a file descriptor of the socket, return 0 if success, -1 if error

\*/

close(socket\_listener);

}

return 0;

}

## 2.2. subserver.cpp

/\*

\* not for launch from terminal

\* launches, when forks from main server

\*/

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <algorithm>

#include <ctime>

using namespace std;

typedef struct timeval TimeValue;

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

{

// ---------- INITIALIZING & PREPARING ----------

int local\_buffer\_length; // length (size) of local buffer

int local\_socket\_fd = atoi(argv[1]); // client's socket handle

char local\_symbol = '0'; // additional variable to save old number in sorting algorithm ('0' here just for fun)

char local\_consequence\_buffer[10]; // number consequence local buffer

// ---------- RECIEVING MESSAGE FROM CLIENTS & SEQUENCE TREATMENT & SENDING MESSAGE TO CLIENTS ----------

// reading a message from client (which was in server, but we forked process, so we have it there)

// then we sorting it out (from unordered numbers to ordered numbers 0..9)

// after that we sending it back (we recieved as ARGUMENT a client's address,

// where the message was from and where we need to send it after treatment)

TimeValue time\_value;

time\_value.tv\_sec = 5;

time\_value.tv\_usec = 0;

setsockopt(local\_socket\_fd, SOL\_SOCKET, SO\_RCVTIMEO, (const char\*)&time\_value, sizeof(time\_value)); // setting a parameters for socket-reciever

/\*

https://man7.org/linux/man-pages/man2/recv.2.html

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags)

recieves a message (writes in buffer) from socket w/ concrete length (size), return the number of bytes recieved if success, -1 if error

\*/

local\_buffer\_length = recv(local\_socket\_fd, local\_consequence\_buffer, 10, 0); // message recieving

cout << "---------- MESSAGE HAS BEEN RECIEVED ----------\n"

<< "---------- BEGIN MESSAGE ----------\n"

<< local\_consequence\_buffer << "\n" // output UNtreated message from client to server to subserver (this) to client

<< "---------- END MESSAGE ----------\n";

// by the 3rd exercice we need to get (server) number consequence

// sort it, and then return (to client) the ordered number consequence

// so we have there a classic sorting algorithm to make from

// unordered consequence the ordered consequence of numbers

// e.g. "4278600937" --> "0023467789"

for (int i = 0; i < 10; i++)

{

for (int j = 0; j < 9; j++)

{

// https://stackoverflow.com/questions/5029840/convert-char-to-int-in-c-and-c

if ((local\_consequence\_buffer[j] - '0') > (local\_consequence\_buffer[j + 1] - '0'))

{

local\_symbol = local\_consequence\_buffer[j];

local\_consequence\_buffer[j] = local\_consequence\_buffer[j + 1];

local\_consequence\_buffer[j + 1] = local\_symbol;

}

}

}

/\*

https://man7.org/linux/man-pages/man2/send.2.html

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags)

send message (located in buffer) on a socket w/ concrete legnth (size), return the number of bytes sent if success, -1 if error

\*/

if (send(local\_socket\_fd, local\_consequence\_buffer, local\_buffer\_length, 0) > 0)

{

cout << "---------- MESSAGE HAS BEEN SENDED ----------\n"

<< "---------- BEGIN MESSAGE ----------\n"

<< local\_consequence\_buffer << "\n" // output treated message from client to server to subserver (this) to client

<< "---------- END MESSAGE ----------\n\n";

}

else

{

cout << "---------- MESSAGE SENDING TO CLIENT ERROR ----------\n\n";

}

// ---------- CLEANING & TERMINATING ----------

/\*

https://man7.org/linux/man-pages/man2/close.2.html

int close(int fd)

closing a file descriptor of the socket, return 0 if success, -1 if error

\*/

close(local\_socket\_fd);

exit(0);

}

## 2.3. client.cpp

/\*

\* ./client

\*

\* could be many clients

\*

\*/

#include <iostream>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <ctime>

using namespace std;

typedef struct sockaddr\_in SocketAddressIn; // struct with server address w/ IP

typedef struct sockaddr SocketAddress; // struct with server address

typedef struct timeval TimeValue; // struct w/ max waiting time to sending a message

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

{

// ---------- INITIALIZING & PREPARING ----------

int i = 0; // standart variable for loop

int socket\_fd; // client's socket handle

int socket\_detect; // client's socket change

int is\_connected = 0; // is client connected to the socket

int begin\_time = 0; // begin time

char message\_send[10]; // message to send from client (this) to server

char message\_receive[10]; // treated message to recieve from server to client (this)

fd\_set readfds; // set of the handles

SocketAddressIn socket\_address\_in; // struct with server address

TimeValue time\_value; // struct w/ max waiting time to sending a message

// ---------- SOCKET CREATION ----------

// creating socket, sill be sent to the server

// setting the connetion port, protocol IPv4 and address to FIND server

// this settings must be the same in the server part

// https://stackoverflow.com/questions/52801380/srandtimenull-function

srand(time(NULL));

/\*

https://man7.org/linux/man-pages/man2/socket.2.html

int socket(int domain, int type, int protocol)

creates socket (w/ domain in domain, protocol in protocol), return file descriptor for the new socket if successful, -1 if error

AF\_INET -- IPv4

SOCK\_STREAM -- two endpoints bytes stream

\*/

socket\_fd = socket(AF\_INET, SOCK\_STREAM, 0); // socket creation, TCP/IP interaction, steam socket

if(socket\_fd < 0)

{

perror("Socket creation error");

exit(1);

}

socket\_address\_in.sin\_family = AF\_INET; // network interaction, IPv4 protocol

socket\_address\_in.sin\_port = htons(3434); // port number in network byte order

socket\_address\_in.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // ip-address; "127.0.0.1" is "localhost"

// ---------- WAITING CONNECTION TO THE SERVER ----------

// establishing a connection w/ server

begin\_time = time(NULL);

cout << "---------- WAITING CONNECTION TO THE SERVER FOR 15 SECONDS ----------\n";

/\*

https://man7.org/linux/man-pages/man2/connect.2.html

int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen)

initiates connection w/ socket, w/ socket address & address length (size), return 0 if success, -1 if error

\*/

while ((time(NULL) - begin\_time) < 15

&& (is\_connected = connect(socket\_fd, (SocketAddress\*)&socket\_address\_in, sizeof(socket\_address\_in))) < 0) // waiting time to connect server -- 15 seconds

{}

if (is\_connected == -1)

{

cout << "---------- COULDN'T CONNECT TO THE SERVER ----------\n";

/\*

https://man7.org/linux/man-pages/man2/close.2.html

int close(int fd)

closing a file descriptor of the socket, return 0 if success, -1 if error

\*/

close(socket\_fd);

exit(-1);

}

// ---------- CREATING SEQUENCE & SENDING MESSAGE TO SERVER ----------

// creating random sequence w/ 10 numbers

// setting timeout for treatment in server,

// sending message and output in in terminal

// https://stackoverflow.com/questions/4629050/convert-an-int-to-ascii-character

for (i = 0; i < 10; i++)

{

message\_send[i] = '0' + rand()%10;

}

// https://stackoverflow.com/questions/4629050/convert-an-int-to-ascii-character

// 90... + 10... is for situation, when number is starts w/ 0... and it will be less, than 10 characters

//sprintf(message, "%lu", rand()%9000000000 + 1000000000); // creating random number sequence

// setting waiting time to work in server part

time\_value.tv\_sec = 15;

time\_value.tv\_usec = 0;

/\*

https://man7.org/linux/man-pages/man3/setsockopt.3p.html

int setsockopt(int socket, int level, int option\_name, const void \*option\_value, socklen\_t option\_len)

sets the socket options (written in option\_name) w/ specified protocol (writter in level), return 0 if success, -1 if error

SOL\_SOCKET -- options at the socket level

SO\_SNDTIMEO -- setting timeout for socket (in option\_value & option\_len)

\*/

setsockopt(socket\_fd, SOL\_SOCKET, SO\_SNDTIMEO, (const char\*)&time\_value, sizeof(time\_value)); // setting a parameters for socket before sending

/\*

https://man7.org/linux/man-pages/man2/send.2.html

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags)

send message (located in buffer) on a socket w/ concrete legnth (size), return the number of bytes sent if success, -1 if error

\*/

send(socket\_fd, message\_send, sizeof(message\_send), 0); // sending a message to server

cout << "---------- MESSAGE HAS BEEN SENDED ----------\n"

<< "---------- BEGIN MESSAGE ----------\n"

<< message\_send << "\n" // sended message from client (this) to server output

<< "---------- END MESSAGE ----------\n";

// ---------- RECIEVING MESSAGE FROM SERVER ----------

// waiting a responce from server,

// setting timeout to receive message and output recieved treated (ordered) message

FD\_ZERO(&readfds); // freeing set of the handles (setting it to zero)

FD\_SET(socket\_fd, &readfds); // adding the handle into the set of the handles (adding socket, which was argument in this function)

// setting waiting time to receive responce message from server to client (this)

time\_value.tv\_sec = 15;

time\_value.tv\_usec = 0;

/\*

https://man7.org/linux/man-pages/man2/select.2.html

int select(int nfds, fd\_set \*restrict readfds, fd\_set \*restrict writefds, fd\_set \*restrict exceptfds, struct timeval \*restrict timeout)

void FD\_SET(int fd, fd\_set \*set)

void FD\_ZERO(fd\_set \*set)

monitoring file descriptors to find ready for I/O operations, return number of file descriptors contained in the three returned descriptor sets/0 if timeout if success, -1 if error

\*/

socket\_detect = select(FD\_SETSIZE, &readfds, NULL, NULL, &time\_value); // socket state detection (change) to get message from server to client (this)

// if nothing (no message) -- output timeout

// else (have message) -- output treated message

if(socket\_detect == 0)

{

cout << "---------- MESSAGE RECIEVE TIMEOUT ----------\n";

}

else

{

/\*

https://man7.org/linux/man-pages/man2/recv.2.html

ssize\_t recv(int sockfd, void \*buf, size\_t len, int flags)

recieves a message (writes in buffer) from socket w/ concrete length (size), return the number of bytes recieved if success, -1 if error

\*/

recv(socket\_fd, message\_receive, sizeof(message\_receive), 0); // recieving a message

cout << "---------- MESSAGE HAS BEEN RECIEVED ----------\n"

<< "---------- BEGIN MESSAGE ----------\n"

<< message\_receive << "\n" // treated message from server to client (this) output

<< "---------- END MESSAGE ----------\n";

}

// ---------- CLEANING & TERMINATING ----------

/\*

https://man7.org/linux/man-pages/man2/close.2.html

int close(int fd)

closing a file descriptor of the socket, return 0 if success, -1 if error

\*/

close(socket\_fd);

return 0;

}

# 3. Скриншоты работы каждой программы

Программа-сервер «server» запускается с помощью команды «./server» (программа-сервер может быть запущена только одна). Программа-клиент «client» запускается с помощью команды «./client» (программ-клиентов может быть запущено несколько). Программа «subserver» не запускается пользователем, и предназначена для запуска программой «server».

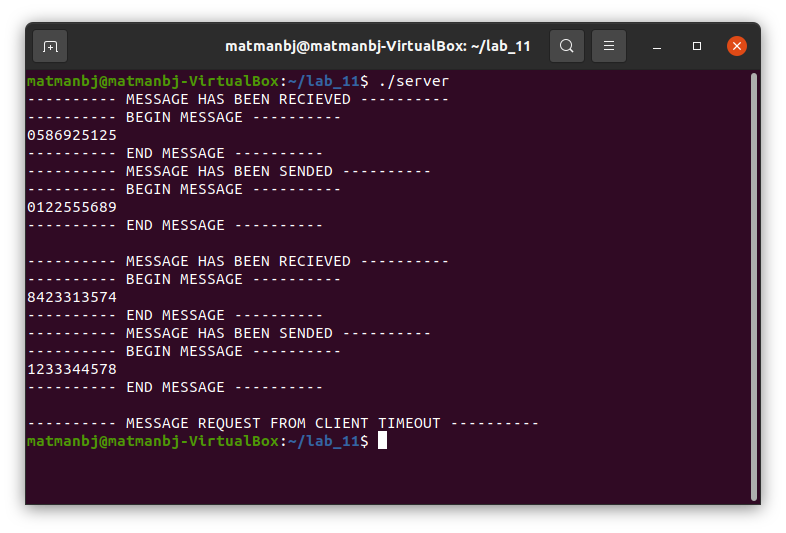


Рисунок 1. Запуск программы-сервера (до запуска программы-клиента 1 и до запуска программы-клиента 2)

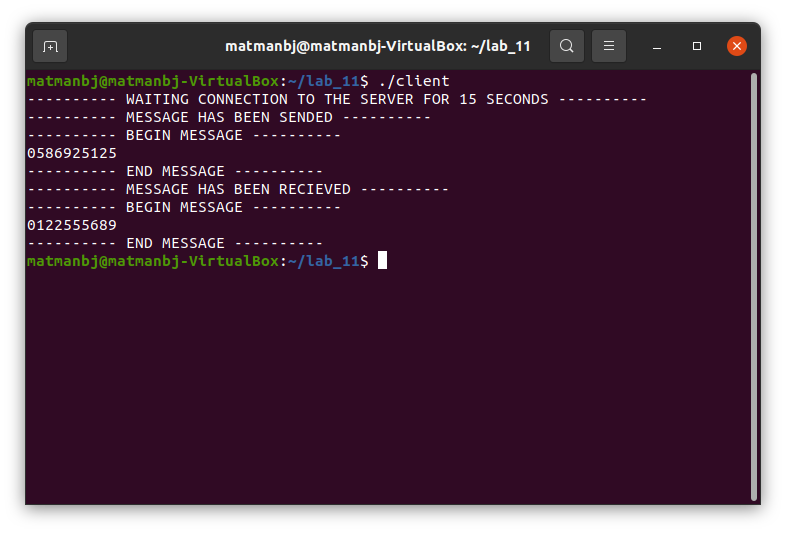


Рисунок 2. Запуск программы-клиента 1 (после запуска программы-сервера и до запуска программы-клиента 2)

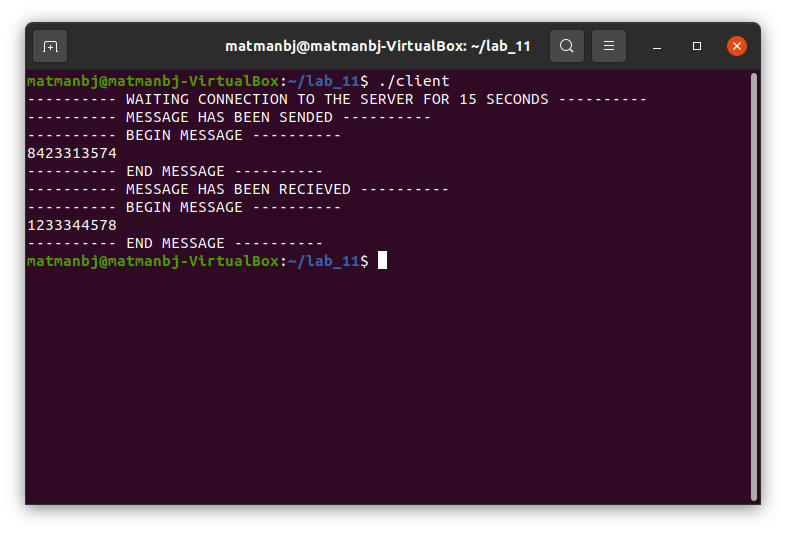


Рисунок 3. Запуск программы-клиента 2 (после запуска программы-сервера и после запуска программы-клиента 1)

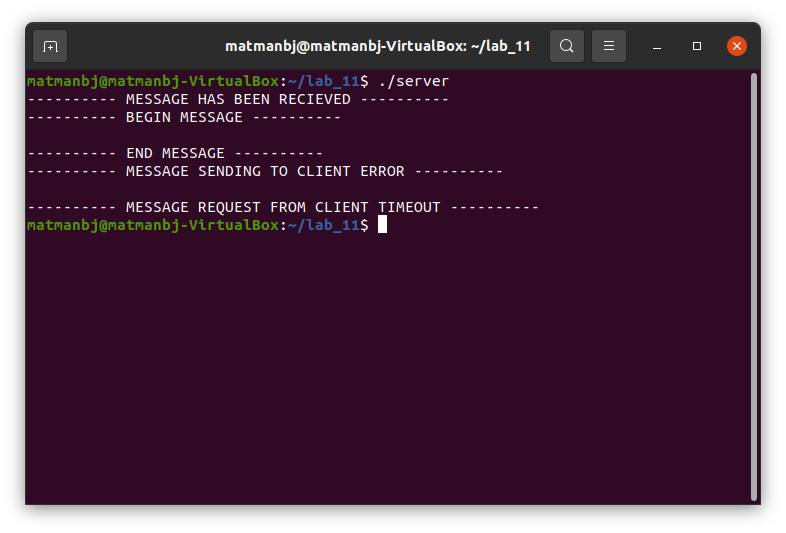


Рисунок 4. Запуск программы-сервера (до запуска программы-клиента с закомментированной строчкой «send» в ней), что приведёт к ошибке времени ожидания по истечении 5-ти секунд

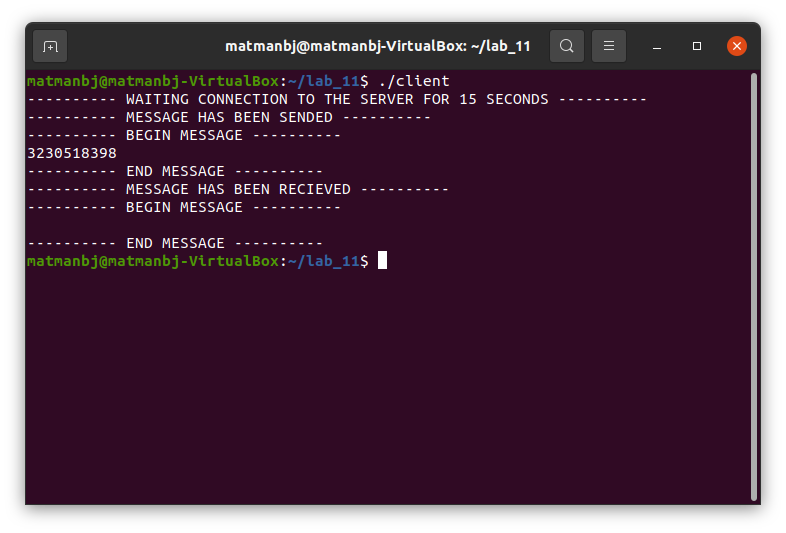


Рисунок 5. Запуск программы-клиента с закомментированной строчкой «send» в ней (после запуска программы-сервера), что приводит к ошибке времени ожидания по истечении 5-ти секунд

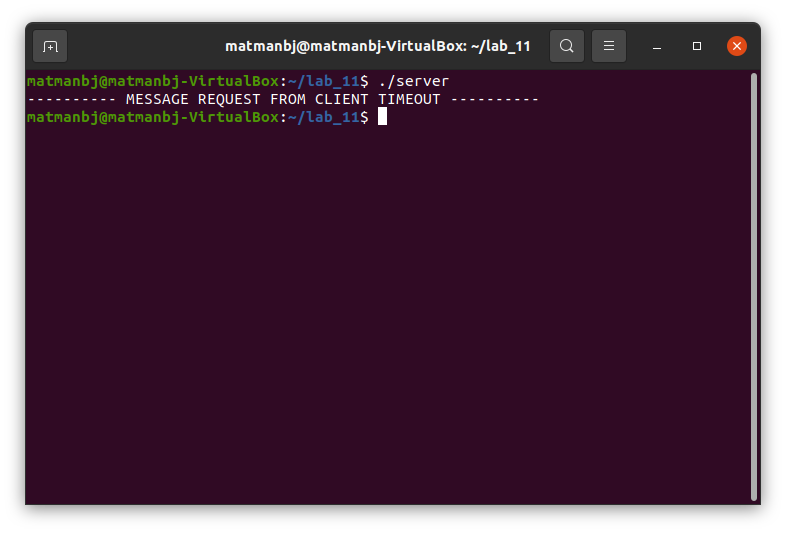


Рисунок 6. Запуск программы-сервера без запуска программы-клиента, что приводит к ошибке времени ожидания по истечении 15-ти секунд

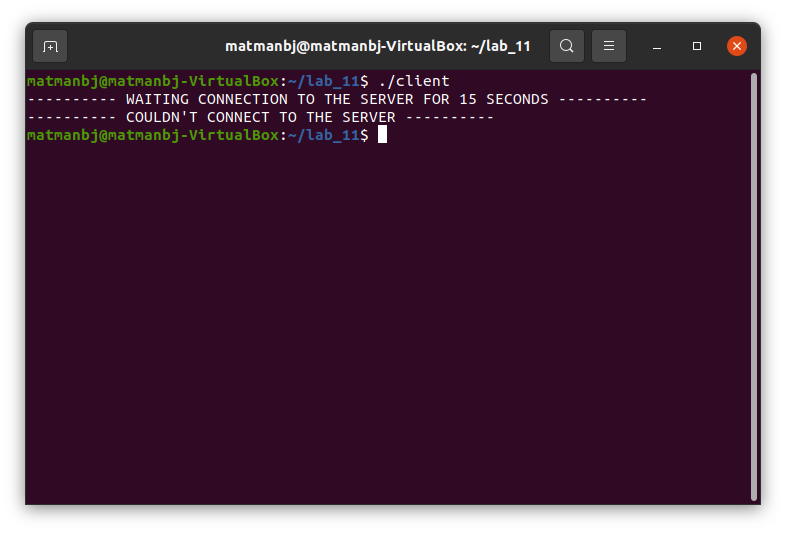


Рисунок 7. Запуск программы-клиента без запуска программы-сервера, что приводит к ошибке времени ожидания по истечении 15-ти секунд

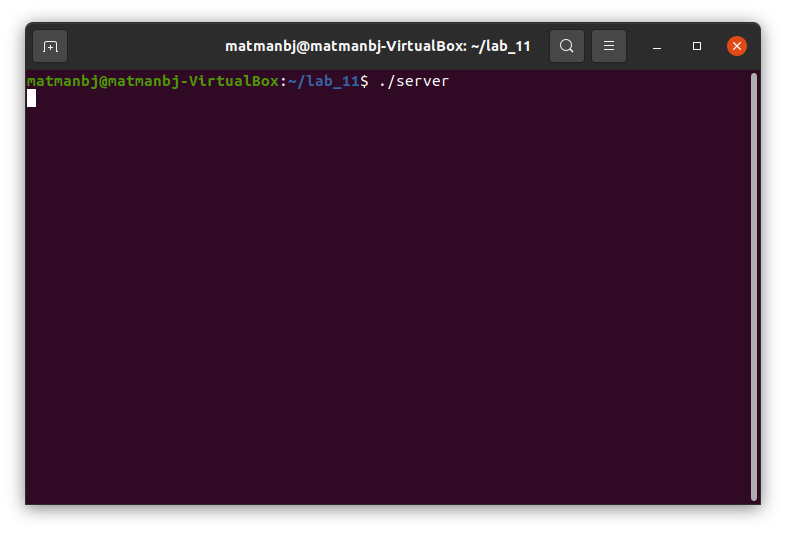


Рисунок 8. Запуск программы-сервера 1 (до запуска программы-сервера 2), что приведёт к ошибке привязки к сетевому адресу

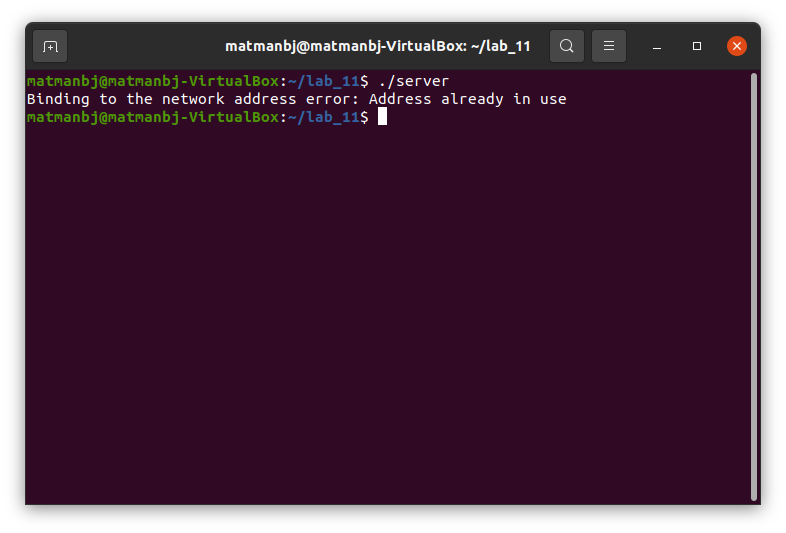


Рисунок 9. Запуск программы-сервера 2 (после запуска программы-сервера 1), что приводит к ошибке привязки к сетевому адресу

# 4. Вывод

В ходе выполнения лабораторной работы №11 «Взаимодействие процессов через сокеты» были созданы программы сервера, подсервера и клиента, отвечающие за приём, обработку и отправку запросов (сообщений) соотвественно. От одного до нескольких клиентов отправляли запрос (сообщение) на сервер в виде неупорядоченной последовательности чисел, затем сервер принимал эти запросы (сообщения), передавал их подсерверу, чтобы он их прочёл, обработал и отправил обратно в виде упорядоченной последовательности. Таким образом и было произведено знакомство с механизмом взаимодействия процессов через сокеты.

# 5. Список использованных источников

1. Онлайн-курс «Организация процессов и программирование в среде Linux» в LMS Moodle [сайт]. URL: <https://vec.etu.ru/moodle/course/view.php?id=9703>.

2. Разумовский Г.В. Организация процессов и программирование в среде Linux: учебно-методическое пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 40с.