Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Операционные среды и системное программирование»

|  |  |
| --- | --- |
|  |  |

**ОТЧЕТ**

к лабораторной работе № 6

на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

Выполнил             В. Д. Ключинский

Проверил                          Н. Ю. Гриценко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc164952590)

[2 Краткие теоретические сведения 4](#_Toc164952591)

[3 Полученные результаты 7](#_Toc164952592)

[Выводы 9](#_Toc164952593)

[Список использованных источников 10](#_Toc164952594)

[Приложение А](#_Toc164952595) [(обязательное)](#_Toc164952596) [листинг исходного кода 11](#_Toc164952597)

**1 ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является практическое освоение основ построения и функционирования сетей, стеков протоколов, программных интерфейсов. Изучение сетевой подсистемы и программного интерфейса сокетов в unix-системах. практическое проектирование, реализация и отладка программ, взаимодействующих через сеть TCP/IP.

# **2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сокет является фундаментальной единицей всего сетевого программирования в Linux (и большинстве других операционных систем). Так же, как функции файлового ввода-вывода определяют интерфейс взаимодействия с файловой системой, сокет соединяет программу с сетью. С его помощью она посылает и принимает сообщения.

Библиотека функций работы с сокетами – Socket API (Application Programming Interface) – является основным инструментом программиста при создании сетевых приложений. Socket API был впервые реализован в операционной системе Berkley UNIX. Сейчас этот программный интерфейс доступен практически в любой модификации Unix, в том числе в Linux. Хотя все реализации чем-то отличаются друг от друга, основной набор функций в них совпадает. Изначально сокеты использовались в программах на C/C++, но в настоящее время средства для работы с ними предоставляют многие языки (Perl, Java и др.). [1]

Сокеты предоставляют весьма мощный и гибкий механизм межпроцессного взаимодействия (IPC, InterProcess Communication). Они могут использоваться для организации взаимодействия программ на одном компьютере, по локальной сети или через Internet, что позволяет создавать распределённые приложения различной сложности. Кроме того, с их помощью можно организовать взаимодействие с программами, работающими под управлением других операционных систем. Например, под Windows существует интерфейс Window Sockets, спроектированный на основе socket API.

Сокеты поддерживают многие стандартные сетевые протоколы (конкретный их список зависит от реализации) и предоставляют унифицированный интерфейс для работы с ними. Наиболее часто сокеты используются для работы в IP-сетях. В этом случае их можно использовать для взаимодействия приложений не только по специально разработанным, но и по стандартным протоколам – HTTP, FTP, Telnet и т. д. Например, имеется возможность написать собственный Web-браузер или Web-сервер, способный обслуживать одновременно множество клиентов.

Таким образом, сокеты – весьма мощное и удобное средство для сетевого программирования. [2]

Сокет (socket) ­– это конечная точка сетевых коммуникаций. Он является чем-то вроде "портала", через которое можно отправлять байты во внешний мир. Приложение просто пишет данные в сокет; их дальнейшая буферизация, отправка и транспортировка осуществляется используемым стеком протоколов и сетевой аппаратурой. Чтение данных из сокета происходит аналогичным образом.

В программе сокет идентифицируется дескриптором – это просто переменная типа **int**. Программа получает дескриптор от операционной системы при создании сокета, а затем передаёт его сервисам socket API для указания сокета, над которым необходимо выполнить то или иное действие.

Подобно системному вызову open(), создающему дескриптор для доступа к файлам и системным устройствам, функция socket() создает дескриптор, позволяющий обращаться к компьютерам по сети.

С каждым сокетом связываются три атрибута: *домен* (domain), *тип* (type) и *протокол* (protocol). Эти атрибуты задаются при создании сокета и остаются неизменными на протяжении всего времени его существования. Для создания сокета используется функция **socket**, имеющая следующий прототип.

Домен определяет пространство адресов, в котором располагается сокет, и множество протоколов, которые используются для передачи данных. Чаще других используются домены Unix и Internet, задаваемые константами **AF\_UNIX** и **AF\_INET** соответственно (префикс AF означает "address family" – "семейство адресов"). При задании **AF\_UNIX** для передачи данных используется файловая система ввода/вывода Unix. В этом случае сокеты используются для межпроцессного взаимодействия на одном компьютере и не годятся для работы по сети. Константа **AF\_INET** соответствует Internet-домену. Сокеты, размещённые в этом домене, могут использоваться для работы в любой IP-сети. Существуют и другие домены (**AF\_IPX** для протоколов Novell, **AF\_INET6** для новой модификации протокола IP - IPv6 и т. д.). [3]

Иногда вместо префикса AF\_ используется префикс PF\_ ("protocol family"). Тип сокета определяет способ передачи данных по сети. Чаще других применяются:

**SOCK\_STREAM**. Передача потока данных с предварительной установкой соединения. Обеспечивается надёжный канал передачи данных, при котором фрагменты отправленного блока не теряются, не переупорядочиваются и не дублируются. Поскольку этот тип сокетов является самым распространённым, до конца раздела мы будем говорить только о нём. Остальным типам будут посвящены отдельные разделы.

**SOCK\_DGRAM**. Передача данных в виде отдельных сообщений (датаграмм). Предварительная установка соединения не требуется. Обмен данными происходит быстрее, но является ненадёжным: сообщения могут теряться в пути, дублироваться и переупорядочиваться. Допускается передача сообщения нескольким получателям (multicasting) и широковещательная передача (broadcasting).

**SOCK\_RAW**. Этот тип присваивается низкоуровневым (т. н. "сырым") сокетам. Их отличие от обычных сокетов состоит в том, что с их помощью программа может взять на себя формирование некоторых заголовков, добавляемых к сообщению.

Обратите внимание, что не все домены допускают задание произвольного типа сокета. Например, совместно с доменом Unix используется только тип **SOCK\_STREAM**. С другой стороны, для Internet-домена можно задавать любой из перечисленных типов. В этом случае для реализации **SOCK\_STREAM** используется протокол TCP, для реализации **SOCK\_DGRAM** - протокол UDP, а тип **SOCK\_RAW** используется для низкоуровневой работы с протоколами IP, ICMP и т. д.

Наконец, последний атрибут определяет протокол, используемый для передачи данных. Как мы только что видели, часто протокол однозначно определяется по домену и типу сокета. В этом случае в качестве третьего параметра функции **socket** можно передать 0, что соответствует протоколу по умолчанию. Тем не менее, иногда (например, при работе с низкоуровневыми сокетами) требуется задать протокол явно. Числовые идентификаторы протоколов зависят от выбранного домена; их можно найти в документации.

Прежде чем передавать данные через сокет, его необходимо связать с адресом в выбранном домене (эту процедуру называют именованием сокета). Иногда связывание осуществляется неявно (внутри функций **connect** и **accept**), но выполнять его необходимо во всех случаях. Вид адреса зависит от выбранного вами домена. В Unix-домене это текстовая строка - имя файла, через который происходит обмен данными. В Internet-домене адрес задаётся комбинацией IP-адреса и 16-битного номера порта. IP-адрес определяет хост в сети, а порт - конкретный сокет на этом хосте. Протоколы TCP и UDP используют различные пространства портов.

Установка соединения на стороне сервера состоит из четырёх этапов, ни один из которых не может быть опущен. Сначала сокет создаётся и привязывается к локальному адресу. Если компьютер имеет несколько сетевых интерфейсов с различными IP-адресами, вы можете принимать соединения только с одного из них, передав его адрес функции **bind**. Если же вы готовы соединяться с клиентами через любой интерфейс, задайте в качестве адреса константу **INADDR\_ANY**. Что касается номера порта, вы можете задать конкретный номер или 0 (в этом случае система сама выберет произвольный неиспользуемый в данный момент номер порта).

На следующем шаге создаётся очередь запросов на соединение. При этом сокет переводится в режим ожидания запросов со стороны клиентов. Всё это выполняет функция **listen**.

Функция **accept** создаёт для общения с клиентом *новый* сокет и возвращает его дескриптор. Параметр **sockfd** задаёт слушающий сокет. После вызова он остаётся в слушающем состоянии и может принимать другие соединения. В структуру, на которую ссылается **addr**, записывается адрес сокета клиента, который установил соединение с сервером. В переменную, адресуемую указателем **addrlen**, изначально записывается размер структуры; функция **accept** записывает туда длину, которая реально была использована. Если вас не интересует адрес клиента, вы можете просто передать NULL в качестве второго и третьего параметров.

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы была создан групповой чат с централизованной архитектурой (выделенный процесс-сервер и процессы-клиенты). Данная программа была создана средствами языка C++ и скомпилирована с помощью makefile.

Результаты работы программы и исходные файлы представлены на рисунках ниже.

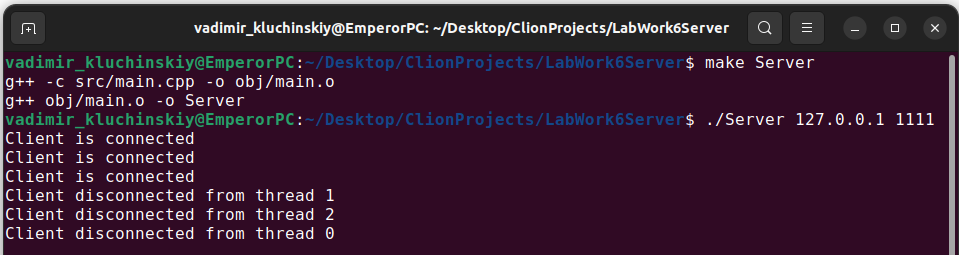


Рисунок 3.1 – Процесс создания сервера и подключения клиентов

На следующем рисунке продемонстрированы сообщения, отправленные пользователем текущим пользователеми полученные сообщения от других пользователей:

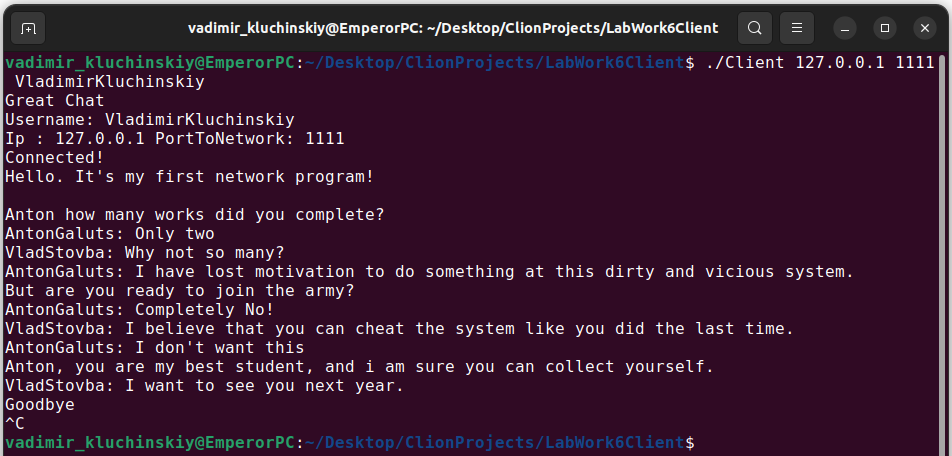


Рисунок 3.2 – Отображение диалога для пользователя VladimirKluchinskiy

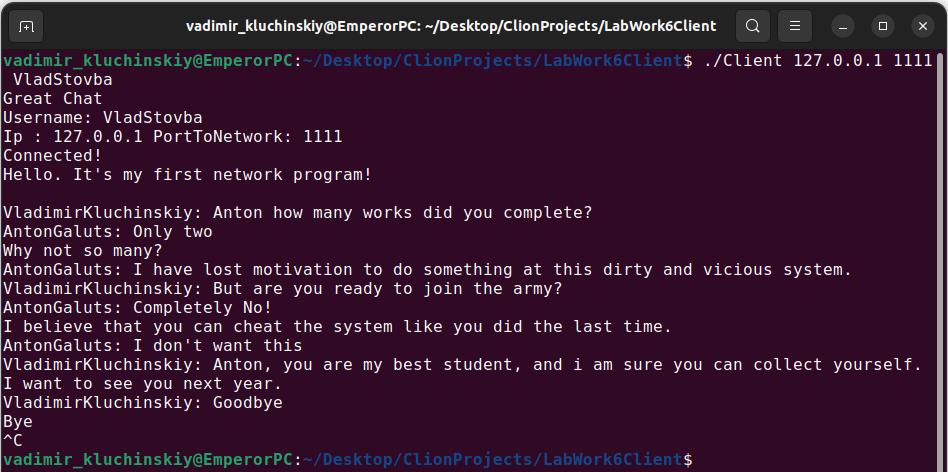


Рисунок 3.3 – Отображение диалога для пользователя VladStovba

При невозможности подключения к серверу выводится соответствующее сообщение, что видно на рисунке 3.4:

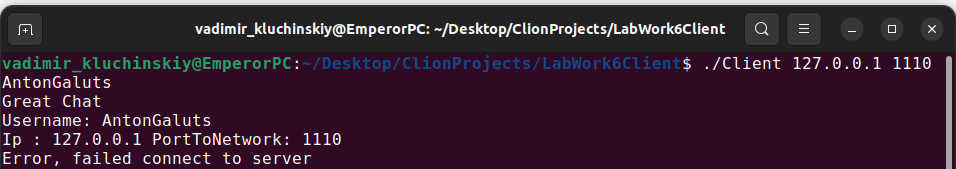


Рисунок 3.4 – Сообщение об ошибке при указании неверного адреса сервера

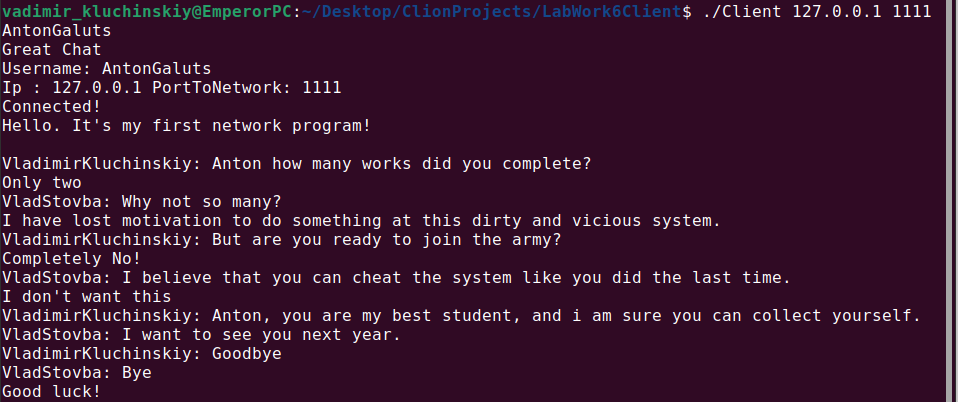


Рисунок 3.5 – Отображение диалога для пользователя AntonGaluts

# **ВЫВОДЫ**

В ходе выполнения данной лабораторный работы была создана программа-чат для интерактивного взаимодействия пользователей, которая имеет централизованную архитектуру. Сервер обрабатывает сигналы от каждого клиента в отдельном потоке. Был изучен процесс отправки и получения информации через сокеты по протоколу TCP, а также атрибуты сокета, установка соединения, закрытие сокета.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Тейнсли, Д. Linux и другие UNIX-подобные операционные системы: программирование. / Д. Тейнсли. – СПб.: БВХ-Петербург, 2006. – 1056 с.

[2] Щупаков, Ю. Руководство по командам и shell-программированию, операционная система Linux. / Ю. Щупаков. – СПб: Питер, 2008. – 592 с.

[3] UnixTutorials [Электронный ресурс]. – Режим доступа: https://prog /unix-tutorials/. – Дата доступа: 06.02.2024.

# **ПРИЛОЖЕНИЕ А**

# **(обязательное)**

# **Листинг исходного кода**

Листинг 1 – Программный код разработанного приложения:

Файл *server.cpp*:

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <thread>

#define MAX\_CONNECTIONS 100

int Connections[MAX\_CONNECTIONS];

int Counter = 0;

std::string Ip;

unsigned short int PortToNetwork;

void Clienthandler(int index) {

int msg\_size;

while (true) {

int recvResult = recv(Connections[index], (char\*)&msg\_size, sizeof(int), 0);

if (recvResult <= 0) {

std::cout << "Client disconnected from thread " << index << std::endl;

close(Connections[index]);

Connections[index] = -1;

return;

}

char\* msg = new char[msg\_size + 1];

msg[msg\_size] = '\0';

int recvResult1 = recv(Connections[index], msg, msg\_size, 0);

if (recvResult1 <= 0) {

std::cout << "Client disconnected from thread " << index << std::endl;

close(Connections[index]);

Connections[index] = -1;

return;

}

for (int i = 0; i < Counter; ++i) {

if (i != index && Connections[i] >= 0) {

send(Connections[i], (char\*)&msg\_size, sizeof(int), 0);

send(Connections[i], msg, msg\_size, 0);

}

}

delete[] msg;

}

}

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

if (argc > 2) {

Ip = argv[1];

PortToNetwork = std::strtoul(argv[2], nullptr, 10);

} else {

Ip = "127.0.0.1";

PortToNetwork = 1111;

}

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

if (sListen == -1) {

std::cerr << "Error creating socket" << std::endl;

return 1;

}

struct sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(PortToNetwork);

serverAddr.sin\_addr.s\_addr = inet\_addr(Ip.c\_str());

memset(serverAddr.sin\_zero, '\0', sizeof serverAddr.sin\_zero);

if (bind(sListen, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) == -1) {

std::cerr << "Bind failed" << std::endl;

return 1;

}

if (listen(sListen, SOMAXCONN) == -1) {

std::cerr << "Listen failed" << std::endl;

return 1;

}

struct sockaddr\_in clientAddr;

socklen\_t clientAddrSize = sizeof(clientAddr);

int newConnection;

while (true) {

newConnection = accept(sListen, (struct sockaddr\*)&clientAddr, &clientAddrSize);

if (newConnection == -1) {

std::cerr << "Accept failed" << std::endl;

} else {

std::cout << "Client is connected" << std::endl;

std::string msg = "Hello. It's my first network program!\n";

int msg\_size = msg.size();

send(newConnection, (char\*)&msg\_size, sizeof(int), 0);

send(newConnection, msg.c\_str(), msg\_size, 0);

Connections[Counter] = newConnection;

++Counter;

std::thread(Clienthandler, Counter - 1).detach();

}

}

close(sListen);

return 0;

}

Файл *client.cpp*:

#include <iostream>

#include <string>

#include <cstring>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <thread>

int Connection;

std::string Ip;

unsigned short int PortToNetwork;

std::string Username;

void ClientHandler() {

int msg\_size;

while (true) {

recv(Connection, (char\*)&msg\_size, sizeof(int), 0);

char\* msg = new char[msg\_size + 1];

msg[msg\_size] = '\0';

recv(Connection, msg, msg\_size, 0);

std::cout << msg << '\n';

delete[] msg;

}

}

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

if (argc > 3) {

Ip = argv[1];

PortToNetwork = std::strtoul(argv[2], nullptr, 10);

Username = argv[3];

} else {

Ip = "127.0.0.1";

PortToNetwork = 1111;

Username = "User";

}

std::cout << "Great Chat\n" << "Username: " << Username << "\nIp : " << Ip << " PortToNetwork: " << PortToNetwork << '\n';

struct sockaddr\_in serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(PortToNetwork);

serverAddr.sin\_addr.s\_addr = inet\_addr(Ip.c\_str());

memset(serverAddr.sin\_zero, '\0', sizeof serverAddr.sin\_zero);

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

if (connect(Connection, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) != 0) {

std::cout << "Error, failed connect to server\n";

exit(1);

}

std::cout << "Connected!\n";

std::thread(ClientHandler).detach();

std::string msg1;

while (true) {

std::getline(std::cin, msg1);

msg1 = Username + ": " + msg1;

int msg\_size = msg1.size();

send(Connection, (char\*)&msg\_size, sizeof(int), 0);

send(Connection, msg1.c\_str(), msg\_size, 0);

usleep(10000); // Подождем 10 миллисекунд

}

return 0;

}